Edition collaborative massive sur réseaux Pair-à Pair

Stéphane Weiss

Nancy-Université

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Edition collaborative massive sur réseaux Pair-à-Pair

THÈSE

présentée et soutenue publiquement le 18 octobre 2010

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Stéphane Weiss

Composition du jury

Président :	Olivier FESTOR	Directeur de Recherche, INRIA Lorraine
Rapporteurs :	Achour MOSTEFAOUI	Maître de Conférences, Université de Rennes
	Marc SHAPIRO	Directeur de Recherche, INRIA Rocquencourt
Examinateur :	Esther PACITTI	Professeur, Université de Montpellier 2
Directeurs de Thèse :	Pascal MOLLI	Professeur, Université de Nantes
	Pascal URSO	Maître de Conférences, Université de Lorraine

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

Loria

Mis en page avec la classe thloria.

Remerciements

Je tiens tout particulièrement à remercier l'ensemble du jury pour l'intérêt porté à mes travaux.

Tout d'abord, je souhaite remercier les rapporteurs Achour Mostefaoui et Marc Shapiro pour leur lecture attentive de mon manuscript et leurs nombreux commentaires qui m'ont permis de l'améliorer.

Je souhaite également remercier Esther Pacitti et Olivier Festor pour leur rôle d'examinateur. Je tiens, de plus, à remercier Olivier Festor pour avoir accepté d'être président du jury.

Je tiens à remercier mes encadrants Pascal Urso et Pascal Molli pour m'avoir guidé tout au long de cette thèse.

À Pascal Urso, pour m'avoir enseigné les rudiments du formalisme qui faisaient cruellement défaut à ma formation initiale. Je le remercie pour les nombreuses et toujours passionnantes discussions que nous avons eu.

À Pascal Molli, pour m'avoir dirigé lors de ma thèse. Je le remercie pour sa vision éclairée du web, ses nombreux conseils et critiques qui m'ont permis de progresser.

Je tiens aussi à remercier Claude Godart pour m'avoir permis de découvrir le monde de la recherche et de l'enseignement.

Je ne pourrais pas finir sans remercier tous ceux qui m'ont accompagné durant ces années, je pense bien sûr aux membres de l'équipe SCORE et tout particulièrement à Gérald Oster, Claudia Ignat, Charbel Rahhal, Nawal Guermouche, Hien Truong mais aussi à Ioana Ilea, Guzman Llambias, Victor Munteanu.

Finalement, je souhaite remercier ma famille qui, par son soutien et sa présence, a contribué à cette thèse.

Table des matières

Table des figures

Introduction

En 2003, Dale Dougherty, co-fondateur de O'Reilly Media, proposa le terme « Web 2.0 » pour désigner une nouvelle étape dans l'évolution du Web : le Web devient *collaboratif*. Un système collaboratif implique plusieurs utilisateurs dans la réalisation d'une tâche commune. Cette dernière peut être, par exemple, la rédaction d'un document. On parle alors, d'édition collaborative.

Les éditeurs collaboratifs se basent sur la réplication afin d'obtenir une grande réactivité [SJZ+98] : Un système d'édition collaborative distribué est composé d'un nombre connu ou inconnu de répliques. Une réplique peut être modifiée du document à chaque instant. Chaque modification est alors envoyée *sous forme d'une opération* aux autres répliques. Cette propagation est supposée fiable : une opération est inévitablement reçue par toutes les répliques. Lorsqu'une réplique reçoit une opération, elle l'exécute.

Un système d'édition collaborative est correct s'il respecte le modèle CCI [SJZ+98] :

- *préservation de la causalité :* l'exécution des opérations doit respecter la relation de précédence définie par Lamport [Lam78],
- *convergence :* toutes les répliques ont le même contenu après avoir reçu toutes les opérations,
- *préservation de l'intention :* L'effet de l'exécution d'une opération doit être le même sur toutes les répliques, et son exécution ne doit pas modifier l'effet d'opérations indépendantes.

De plus, l'annulation est une fonctionnalité essentielle des éditeurs collaboratifs et mono-utilisateur. Dans un système d'édition collaborative, le mécanisme d'annulation doit permettre l'annulation de n'importe quelle modification. L'ajout de cette fonctionnalité ne doit, bien sûr, pas compromettre la vérification du modèle de cohérence CCI. Fournir un tel mécanisme d'annulation est un problème difficile [Sun00].

L'arrivée du Web 2.0 implique des conséquences sur la conception d'un système d'édition collaborative : l'édition collaborative devient massive. Par exemple, Wikipédia est un système d'édition collaborative dans lequel n'importe quel utilisateur peut contribuer. Ce système a permis d'obtenir 15 millions d'articles en seulement 9 ans [Wik10a].

Il est donc nécessaire de concevoir des systèmes collaboratifs capables de gérer la charge induite par un nombre massif d'utilisateurs. Afin de soutenir cette charge, de nombreux systèmes collaboratifs se tournent vers une architecture dite pair-à-pair.

Ces systèmes sont construits pour permettre le passage à l'échelle, l'auto-organisation,

etc. Afin d'atteindre ces objectifs, ils doivent passer à l'échelle sur les points suivants :

- *La dynamicité :* Un système pair-à-pair passe à l'échelle en termes de dynamicité, s'il tolère de grandes variations du nombre de répliques ainsi que de la topologie du réseau.
- *La symétrie :* Dans un système pair-à-pair, il peut exister des points centraux, c'est-à-dire, des répliques dont le non-fonctionnement empêche le fonctionnement d'autres répliques dépendantes. Un système pair-à-pair passe à l'échelle en termes de décentralisation s'il possède un nombre réduit de répliques dépendantes. Plus un système est symétrique, plus il résiste aux pannes.
- *Le nombre massif d'utilisateurs et de données :* Le système doit tolérer l'ajout d'utilisateurs ou d'objets sans subir une perte de performance notable [Neu94].

Parmi les systèmes existants, certains vérifient le modèle de cohérence CCI mais ne supportent pas les contraintes des réseaux pair-à-pair. D'un autre côté, il existe de nombreux systèmes supportant les contraintes pair-à-pair, mais qui ne respectent pas le modèle de cohérence requis pour un système d'édition collaborative.

Le but de cette thèse est donc de fournir un algorithme de réplication capable de supporter les contraintes des réseaux pair-à-pair tout en respectant le niveau de cohérence requis pour un système d'édition collaborative.

Dans ce manuscrit, nous proposons :

- Un *modèle formel* pour les systèmes d'édition collaborative, ce qui nous permet de formaliser le modèle CCI, y compris la préservation de l'intention dans le cas d'un système d'édition collaborative de texte avec annulation.
- *Logoot*, un algorithme appartenant à la classe d'algorithmes « type de données répliqué commutatif » (CRDT) pour la conception de système d'édition collaborative de texte. L'idée principale de cette approche est d'associer à chaque éléments du document un identifiant unique. L'ensemble de ces identifiants est totalement ordonné et dense, ce qui permet d'exécuter les opérations en temps logarithmique en utilisant l'algorithme de recherche dichotomique. Nous montrons que notre approche vérifie bien le modèle CCI. De plus, nous avons réalisé des expérimentations afin d'évaluer le passage à l'échelle de notre approche. Cette contribution a été validée par la publication d'un article dans la conférence ICDCS [WUM09].
- *CRDT*$^+$, un type de données CRDT permettant l'annulation. Couplé à un système d'édition collaborative basé sur un CRDT, on obtient un nouveau système d'édition collaborative possédant une fonctionnalité d'annulation. Nous montrons que si le système initial vérifie une propriété appelée *Inversibilité*, le résultat est aussi un CRDT possédant un mécanisme d'annulation correct.
- *Logoot*$^+$ est la résultante de l'application du CRDT$^+$ sur Logoot. Nous avons modifié Logoot afin qu'il vérifie les propriétés requises. Nous obtenons donc une version de l'algorithme Logoot étendue avec la fonctionnalité d'annulation. De nouvelles expérimentations ont été menées afin de montrer le passage à l'échelle de cette

proposition. Cette approche a été publiée dans la revue TPDS [WUM10].

Chapitre 1

Problématique

Sommaire

1.1 Les systèmes collaboratifs

En 2003, Dale Dougherty, co-fondateur de O'Reilly Media, proposa le terme « Web 2.0 » pour désigner une nouvelle étape dans l'évolution du Web. D'après Tim O'Reilly et John Battelle [OB09] :

> « Le Web 2.0 est tout ce qui concerne l'exploitation de l'intelligence collective. »

L'exploitation de l'intelligence collective est souvent associée au terme de « crowdsourcing » [OB09] que l'on peut traduire par « externalisation à grande échelle » [Cro10]. En effet, dans le Web 2.0, le contenu d'un service n'est pas généré par le fournisseur du service mais par les utilisateurs. Par exemple, Wikipédia [Wik], une encyclopédie en ligne autorise l'édition de ses articles par n'importe quel utilisateur. Le Web 2.0 est donc un *Web collaboratif*. Ellis *et al.* [EGR91] définissent les systèmes collaboratifs comme :

> « des systèmes basés sur des ordinateurs qui soutiennent des groupes de personnes impliquées dans une tâche commune (ou un objectif commun) et qui fournissent une interface à un environnement partagé. »

Cette tâche commune peut être la rédaction d'un document, la planification d'une réunion, etc. Par exemple, un système de messagerie, instantanée ou non, peut servir à fixer un rendez-vous entre deux personnes ou plus. L'ensemble des messages échangés constitue alors l'espace partagé et le but commun est la détermination d'une date pour la réunion.

Afin de réaliser cette tâche, les utilisateurs vont, suivant le système, devoir être présents ou non au même endroit et au même moment. Ainsi, les systèmes collaboratifs peuvent être catégorisés suivant une classification espace-temps initialement proposée par Johansen [Joh88] puis reprise, entre autres, par Ellis [EGR91]. Cette classification considère deux axes :

- Axe du temps : les systèmes sont classés selon le moment des interactions. Si les interactions entre les utilisateurs doivent avoir lieu au même moment, comme pour le téléphone, ou une réunion, le système appartient à la catégorie « Temps identiques ». Au contraire, si les interactions ne peuvent se faire au même moment, le système appartient à la catégorie « Temps différents ». C'est notamment le cas des systèmes dits « tour à tour ».
- Axe de l'espace : les systèmes sont classés selon le lieu des interactions. Cette classification fait la distinction entre les systèmes requérant que l'interaction ait lieu au même endroit et ceux qui autorisent une interaction à distance.

La figure 1.1 illustre cette classification. Elle comporte quatre catégories que nous présentons brièvement et nous donnons un exemple pour chaque catégorie de cette matrice.

(a) Dans cette catégorie se trouvent tous les systèmes permettant à des utilisateurs d'interagir dans la même pièce au même moment. Un exemple de système pour cette catégorie peut être une console de jeux vidéo. En effet, les utilisateurs sont dans la même pièce et interagissent au même moment avec le même système.

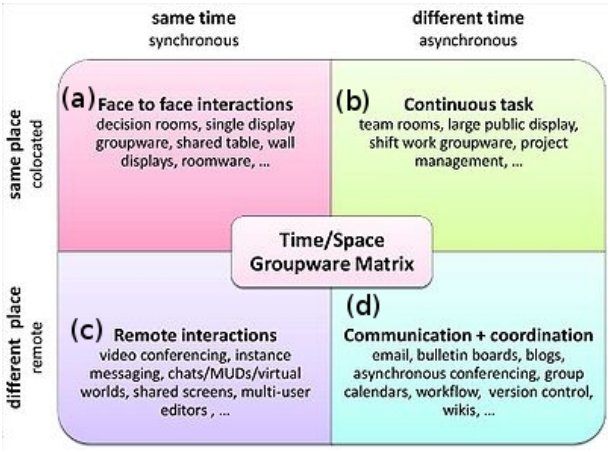


FIGURE 1.1 – Matrice CSCW tirée de [SMIRM07a]

(b) Les systèmes de cette catégorie impliquent des interactions tour à tour [SMIRM07b]. Il peut s'agir par exemple, d'une série d'exposés lors d'une conférence : à un instant donné, un seul intervenant a la parole.

(c) Les systèmes « même temps, lieux différents » incluent le téléphone, les systèmes de vidéo-conférence, les messageries instantanées, etc.

(d) Cette catégorie concerne les systèmes dans lesquels les interactions ont lieu à des endroits et à des moments différents. Il peut s'agir d'une messagerie de type courriel, d'un gestionnaire de configuration [Ber90], d'un wiki, d'un système de « workflow », etc.

Dans ce manuscrit, nous nous intéressons principalement aux systèmes d'édition collaborative qui occupent la case (d) de la matrice figure 1.1 : « Endroits différents, temps différents ».

1.2 Les systèmes d'édition collaborative

Les systèmes d'édition collaborative [EG89] ont pour but la production d'un document. Ce dernier peut être du texte, un fichier XML, une image, une vidéo ou même un répertoire. Les principaux avantages d'un système d'édition collaborative sont l'obtention de meilleures idées et de différents points de vues [NR04]. L'édition collaborative permet aussi de réduire le temps de réalisation d'une tâche. Dans un éditeur collaboratif, chaque utilisateur peut interagir à tout moment avec ce système en modifiant le document ou en annulant une modification précédente [Sun00].

Les éditeurs collaboratifs se basent sur la réplication afin d'obtenir une grande réactivité [SJZ+98]. En effet, chaque action effectuée par un utilisateur distant, implique l'envoi d'un message. En fonction de la distance séparant les utilisateurs, une certaine durée sera nécessaire pour la transmission. Par exemple, la latence du réseau entre Sydney et les

États-Unis est de 70ms [Net10]. Si le document est hébergé par un serveur distant, toute opération nécessite au minimum 140ms pour avoir une réponse du serveur. La réplication permet au contraire de s'adresser à une réplique locale et donc de s'affranchir de la latence du réseau lors de l'édition.

De plus, un système d'édition collaborative doit fournir les différents modes de collaboration [LCL04, SMIRM07a] et en particulier le mode de travail déconnecté. Les éditeurs collaboratifs se basent donc sur des mécanismes de réplication tolérant la déconnexion des répliques.

Dans cette section, nous présentons le modèle des systèmes d'édition collaborative et le modèle de cohérence associé. Puis, nous présentons les problèmes propres à la fonctionnalité d'annulation essentielle à tout éditeur. Finalement, l'ensemble du système doit être adapté à l'édition massive.

1.2.1 Modèle de système d'édition collaborative

Un système d'édition collaborative [EGR91] est composé d'un nombre connu ou inconnu de répliques. Chaque réplique possède une copie du document partagé. On suppose que les répliques tolèrent les pannes. Chaque réplique peut être modifiée à chaque instant : une opération représentant la modification est alors générée, on parle de *génération d'une opération*. Cette opération est alors *exécutée* localement et est envoyée à l'ensemble des répliques. On suppose que la communication entre les répliques est *asynchrone et fiable* : chaque opération est reçue par toutes les répliques. Lorsqu'une réplique reçoit une opération distante, elle l'exécute.

Dans ce manuscrit, nous nous intéressons plus précisément à *l'édition collaborative de documents texte*. Nous considérons les opérations d'insertion et de suppression d'éléments textuels [EG89]. Un élément textuel est généralement un caractère, un mot, une ligne ou un paragraphe. Cependant, on peut aussi considérer une image ou une vidéo comme un élément du document texte.

Modèle de cohérence

Dans cette section, nous détaillons le modèle de cohérence des systèmes d'édition collaborative. Le modèle le plus largement utilisé pour l'édition collaborative est le modèle dit « CCI » [SJZ+98] pour « préservation de la causalité, convergence et préservation de l'intention ». Nous présentons dans cette section ces trois critères.

Préservation de la causalité :

La causalité est un critère qui, s'il est respecté, assure qu'un utilisateur ne verra pas l'effet d'une action dont il n'aurait pas vu la cause au préalable. Le respect de la causalité implique donc une notion de sémantique, puisqu'il faut être en mesure de déterminer quelle cause a produit quel effet.

Par exemple, considérons un éditeur collaboratif de texte grâce auquel trois utilisateurs écrivent un article. Supposons que le premier utilisateur insère une figure numérotée, puis le second la décrit en la référençant. Dans cet exemple, la description dépend de l'insertion de la figure. En effet, si la figure n'avait pas été insérée, le second utilisateur n'aurait pas écrit un paragraphe pour la décrire. Si la causalité est respectée, le troisième utilisateur doit, d'abord voir l'ajout de la figure, puis l'ajout de sa description.

La définition originelle du modèle CCI définit la causalité en tant que relation d'ordre partiel [Lam78] notée $\rightarrow$ et définie par [SJZ+98] :

> « Soient O_a et O_b deux opérations générées sur les répliques i et j. On a $O_a \rightarrow O_b$ [c'est-à-dire, O_a précède causalement O_b] si et seulement si :
>
> 1. $i = j$ et la génération de O_a s'est produite avant la génération de O_b, ou
> 2. $i \neq j$ et O_a a été exécutée sur la réplique j avant la génération de O_b, ou
> 3. Il existe une opération O_x tel que $O_a \rightarrow O_x$ et $O_x \rightarrow O_b$. »

Intuitivement, cette définition spécifie qu'une opération O_a précède causalement une opération O_b si la réplique qui génère O_b a déjà exécuté O_a auparavant. La figure 1.2 illustre le respect de la causalité. Trois répliques répliquent le même document. La première réplique génère O_a, puis l'envoie aux deux autres répliques. Sur la réplique 2, l'opération O_b est générée après réception et exécution de l'opération O_a. L'opération O_a précède donc causalement l'opération O_b.

La causalité peut aussi être définie explicitement, par exemple, Ladin *et al.* [LLSG92] définissent qu'une opération de mise à jour d'un objet dépend des opérations de mise à jour précédentes pour ce même objet. Malheureusement, il n'est pas toujours possible de définir explicitement les relations de dépendance.

Finalement, le modèle CCI définit la préservation de la causalité [SJZ+98] par :

> « La causalité est respectée si pour tout couple d'opérations O_a et O_b, $O_a \rightarrow O_b$ implique que O_a est exécutée avant O_b sur toutes les répliques. »

Cette définition signifie que si une réplique ayant exécuté O_a génère une opération O_b, alors l'ensemble des répliques devra exécuter O_a avant de pouvoir exécuter O_b. Cette définition implique, non seulement de ne pas exécuter pas l'effet avant la cause, mais aussi que toutes les opérations générées soient exécutées par toutes les répliques. Si on reprend l'exemple précédent, afin de respecter la causalité, toutes les répliques doivent donc exécuter l'opération O_a avant l'opération O_b. Par conséquent, la réplique 3, qui a

d'abord reçu O_b, doit retarder son exécution pour attendre la réception de O_a. Une fois O_a exécutée, la réplique 3 peut exécuter O_b.

La causalité doit être préservée non seulement pour assurer le bon fonctionnement du système, mais aussi pour répondre aux exigences imposées par les interactions entre plusieurs utilisateurs [SJZ+98]. Sun *et al.* proposent l'exemple suivant :

> « Supposons qu'un utilisateur sur une réplique 0 pose une question en insérant « Quel jour férié a lieu à Victoria en Australie, le premier mardi de Novembre ? » dans un document texte partagé. Puis l'utilisateur de la réplique 1 répond à cette question en insérant « Melbourne Cup » dans ce même document partagé. L'utilisateur de la réplique 2 serait étonné de voir la réponse avant la question. »

La préservation de la causalité aide donc à la compréhension des actions faites par les autres utilisateurs.

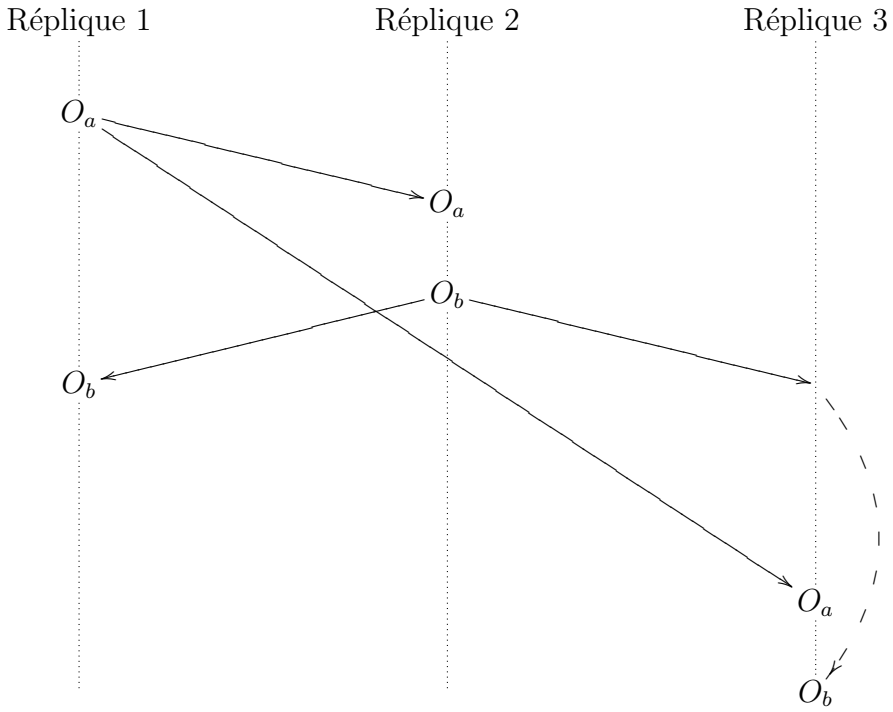


FIGURE 1.2 – Respect de la relation de causalité

Convergence des répliques :

Sun *et al.* [SJZ+98] définissent le critère de convergence par :

> « Lorsque le même ensemble d'opérations a été exécuté sur toutes les répliques, toutes les copies du document partagé sont identiques. »

Par exemple, figure 1.3, les répliques 1, 2 et 3 partagent un document dont la version initiale est V_0. Les répliques 1 et 2 modifient le document, produisant respectivement les

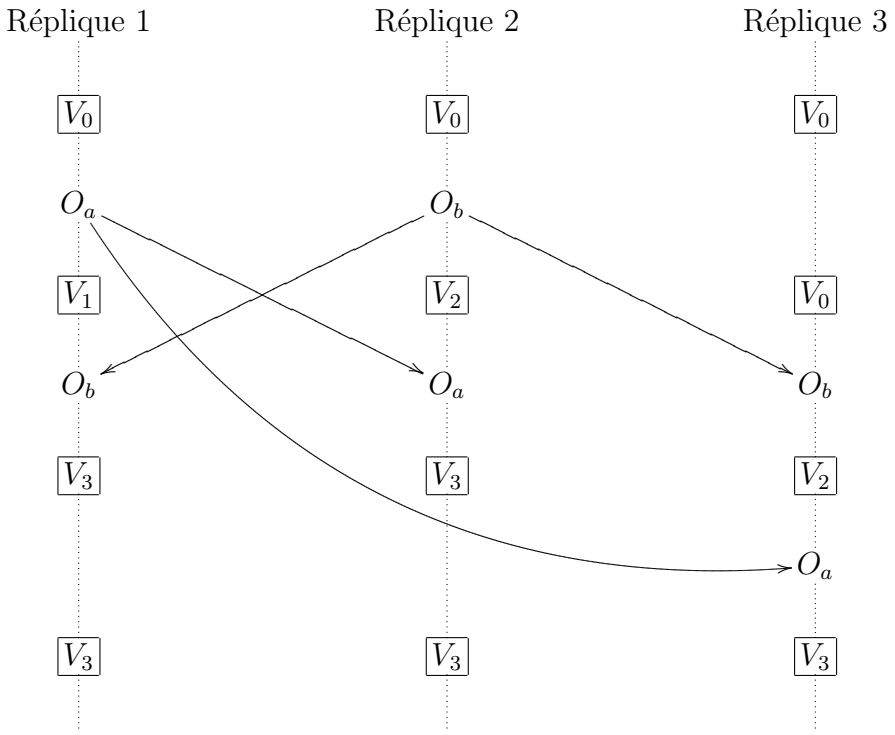


FIGURE 1.3 – Convergence des répliques

opérations O_a et O_b. Après diffusion, réception et exécution de ces opérations par toutes les répliques, la version du document doit être la même sur les trois répliques, ici, la version obtenue est la version V_3.

Préservation de l'intention :

Le critère de convergence impose que toutes les répliques aient la même valeur une fois le système au repos. Ce critère n'impose pas de restriction sur cette valeur. Par exemple, un état de convergence peut être un document vide. Bien évidemment, dans ce cas, la collaboration peut être difficile voire impossible. Il est donc nécessaire d'obtenir des garanties sur l'état de convergence. La *préservation de l'intention* est un critère qui contraint l'état de convergence.

Sun *et al.* [SJZ+98] définissent l'intention d'une opération O par :

> « L'intention d'une opération O est l'effet obtenu par l'application de O sur l'état du document à partir duquel O a été générée. »

L'intention d'une opération est donc l'effet qu'elle produit sur le document lors de sa génération, elle constitue donc le changement souhaité par l'utilisateur. Le critère de préservation de l'intention est respecté si [SJZ+98] :

> « Pour toute opération O, les effets de l'exécution de O sur chaque réplique

> sont équivalents à l'intention de O et, les effets de l'exécution de O ne modifient pas les effets des opérations indépendantes. »

Autrement dit, l'état de convergence doit contenir les modifications de chaque utilisateur. Ce critère est donc subjectif et n'est pas formalisé dans le cas général. Il appartient à chaque système de définir ses intentions et de s'assurer de leur préservation.

Pour un éditeur collaboratif de texte, nous définissons l'intention par la préservation des relations d'ordre partiel entre les éléments du document qui ont été introduites par les insertions (voir section 3.3.3 pour une définition formelle). En plus des relations d'ordre partiel, nous considérons que les opérations ne sont pas affectées par des opérations concurrentes :

- une opération d'insertion d'un élément implique sa présence jusqu'à sa suppression explicite,
- une opération de suppression d'un élément implique qu'il ne soit plus présent dans le document.

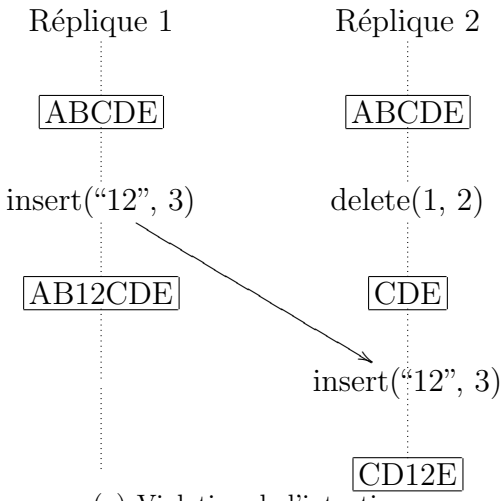
Par exemple, figure 1.4, un système d'édition collaborative de texte partage un document « ABCDE » entre deux répliques. La première réplique insère la chaîne de caractères « 12 » entre les lettres « B » et « C » alors que la seconde supprime les lettres « A » et « B ». Dans un tel scénario, on peut imaginer que le résultat respectant les intentions des utilisateurs est « 12CDE ». Or en représentant l'opération de insertion comme « insérer deux caractères à partir de la position 3 », le système génère le résultat « CD12E » (figure 1.4(a)). Ce résultat ne préserve pas l'ordre partiel induit par l'opération d'insertion. La première réplique souhaite insérer « 12 » entre « B » et « C ». Malheureusement, sur la seconde réplique « 12 » est inséré entre « D » et « E », soit après « C ». Afin de préserver l'intention de la seconde réplique, le système doit insérer les caractères en positions 1 et 2 sur la réplique 2 (figure 1.4(b)).

Remarque (Orthogonalité des critères) : Le modèle de cohérence CCI est composé des trois critères décrits précédemment : préservation de la causalité, convergence et préservation de l'intention. Ces trois critères sont orthogonaux, c.-à-d., la vérification d'un des critères n'est pas requise pour en vérifier un autre. En effet, il existe des systèmes vérifiant uniquement deux de ces critères :

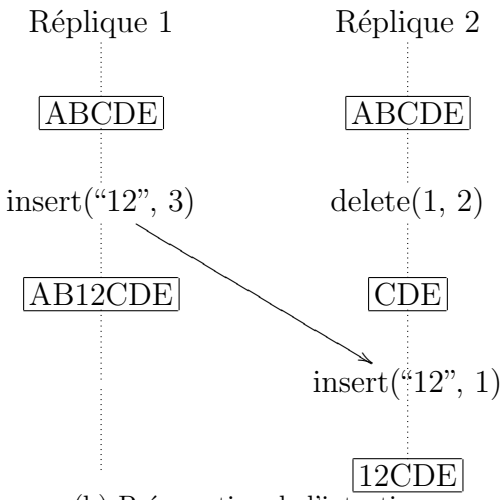
Convergence + Intention : Woot [OUMI06a] est un système dans lequel seul la convergence et les intentions sont vérifiées. En effet, la causalité étant formalisée par des pré-conditions explicites.

Causalité + Intention : Suleiman *et al.* [SCF97] présente un système qui ne vérifie pas la convergence [OUM05] mais qui préserve les intentions.

Causalité + Convergence : Thomas Write Rule [JT75] est une technique permettant d'assurer la convergence sans les autres critères. En effet, bien que ce mécanisme ne permettent pas de voir la cause après l'effet, il est possible d'observer un effet sans jamais en voir la cause. Or la définition de la causalité du modèle CCI[SJZ+98]



(a) Violation de l'intention



(b) Préservation de l'intention

FIGURE 1.4 – Préservation de l'intention

impose d'exécuter cette cause. Si on ajoute une diffusion causale des modifications, on obtient un système n'assurant que la convergence et la causalité.

1.2.2 Annulation dans les systèmes d'édition collaborative

L'annulation est une fonctionnalité essentielle des éditeurs mono-utilisateur. Prakash et Knister [PK94] écrivent :

> « La capacité d'annuler des opérations est une fonction standard et utile dans la plupart des applications interactives mono-utilisateur. Par exemple, la présence d'une fonctionnalité d'annulation dans les éditeurs est utile pour annuler des actions erronées. Elle aide aussi à réduire la frustration des utilisateurs face à de nouveaux systèmes, en particulier ceux qui permettent aux utilisateurs d'appeler des commandes qui peuvent modifier l'état du système de façon com-

> plexe. La présence de l'annulation peut également encourager les utilisateurs à expérimenter, agissant non seulement comme un filet de sécurité, mais aussi en permettant aux utilisateurs d'essayer des approches différentes pour résoudre les problèmes à l'aide du *retour arrière*. »

Cette citation illustre bien l'importance de cette fonctionnalité : elle ne se limite donc pas à simplement annuler des erreurs de manipulation. Elle a un rôle pédagogique en réduisant l'impact d'une erreur. Par exemple, dans certains éditeurs de texte, il est possible de formater l'ensemble du document suivant un ensemble de canevas prédéfinis. Si un utilisateur essaye un format, il perd définitivement sa propre mise en page. L'annulation permet d'annuler cette opération et de revenir à la mise en page de l'utilisateur. L'annulation permet donc à l'utilisateur de tester des fonctionnalités du système. Suivant les éditeurs, on retrouve cette fonctionnalité sous plusieurs formes :

- *Annulation simple* : Seule la dernière modification peut être annulée. On peut généralement annuler cette annulation, c'est-à-dire, rejouer cette modification.
- *Annulation dans l'ordre chronologique inverse* : Les modifications peuvent être annulées dans l'ordre inverse de leur création : de la plus récente à la plus ancienne.
- *Annulation sélective* : l'utilisateur peut choisir les modifications à annuler. L'annulation sélective est la forme la plus générique d'annulation dans le sens où il est possible d'obtenir les autres comportements de l'annulation via la sélection des modifications.

L'annulation de groupe est la transposition de la fonctionnalité d'annulation dans les systèmes d'édition collaborative. L'annulation de groupe est importante pour les mêmes raisons que l'annulation des systèmes mono-utilisateur. Elle présente cependant un autre avantage [PK94] :

> « Si l'état courant du document contient des informations importantes, les personnes peuvent avoir des inhibitions à apporter des modifications parce que ce travail n'est pas uniquement le leur. Savoir que leur modification peut être annulée sans annuler le travail des autres utilisateurs peut aider des membres du groupe à ajouter leurs idées dans le document. »

Elle permet donc de réduire l'appréhension des utilisateurs à participer à la collaboration. Tout comme l'annulation des systèmes mono-utilisateur, l'annulation de groupe existe sous les trois formes décrites précédemment. Elle introduit de plus la notion de *portée* :

- *Locale* : un utilisateur ne peut annuler que ses propres modifications.
- *Globale* : un utilisateur peut annuler les modifications des autres utilisateurs en plus des siennes.

L'annulation de groupe combine deux portées et trois modes d'annulation, soit six différentes configurations. On peut alors se demander si toutes les configurations sont utilisables. Par exemple, Vidot [Vid02] définit les combinaisons possibles de ces portées avec les trois modes d'annulation (voir figure 1.5) dans un système dans lequel les modifications

Portée de l'annulation	Mode d'annulation		
	Simple	Chronologique	Sélective
Locale	Oui	Oui	Oui
Globale	Non	Non	Oui

FIGURE 1.5 – Mode et portée de l'annulation par N. Vidot pour l'édition temps réel [Vid02]

sont intégrées continuellement. Par exemple, dans un tel système, il n'est pas souhaitable d'annuler la dernière modification en portée globale. En effet, l'utilisateur ne connaît pas forcément la dernière modification reçue. Cependant, dans le cas général, l'annulation de groupe doit permettre l'annulation de n'importe quelle modification, effectuée par n'importe quel utilisateur [HM95]. Il est donc nécessaire de fournir une annulation sélective de portée globale.

La principale difficulté de l'annulation de groupe réside dans la gestion de la concurrence [Sun00]. En effet, contrairement à l'annulation des éditeurs mono-utilisateur, l'annulation de groupe doit pouvoir gérer l'annulation de multiples opérations en concurrence (voir figure 1.6), mais aussi l'annulation de la même opération par plusieurs répliques (voir figure 1.7).

Sun [Sun00] définit l'effet de l'annulation par :

> « Soit un document dans l'état :
>
> $$S_n = S_0 \circ [O_1, \ldots, O_{i-1}, O_i, O_{i+1}, \ldots, O_n]$$
>
> [S_n est l'état du document obtenu en exécutant les opérations O_i $(1 \leq i \leq n)$ à partir de l'état initial S_0.] L'effet de l'application de $Annuler(O_i)$ sur l'état S_n est la transformation de l'état S_n en un nouvel état S_{n-1} qui peut être exprimé par
>
> $$S_{n-1} = S_0 \circ [O_1, \ldots, O_{i-1}, O'_{i+1}, \ldots, O'_n]$$
>
> avec O'_x, $x \in \{i+1, i+2, \ldots, n\}$, l'opération que le système aurait exécuté, à la place de O_x, si O_i n'avait pas été exécutée avant. »

D'une façon intuitive, cette définition précise que le système doit retourner dans l'état qu'il aurait atteint, si l'opération à annuler n'avait pas été exécutée. On peut cependant s'interroger sur la capacité du système à calculer l'état S_{n-1}. En effet, les opérations $O_{i+1}, \ldots, O_n$ peuvent dépendre causalement de l'opération O_i.

Les figures 1.6 et 1.7 montrent l'effet attendu de l'annulation conformément à cette définition de l'annulation. Initialement, deux répliques partagent le même document dans l'état initial V_0. Puis, au même moment, la réplique 1 génère O_a pour obtenir une version V_1 et la réplique 2 génère O_b pour obtenir V_2. Puis (figure 1.6) chaque réplique annule l'opération de l'autre réplique. Si les opérations O_a et O_b, n'avaient pas été exécutées, le document serait dans l'état initial, c'est-à-dire, l'état V_0. Par contre, figure 1.7, les deux

répliques annulent la même opération O_b. Cette opération O_b est donc annulée deux fois, et seule l'opération O_a reste « active », donc le résultat doit être V_1. Cette définition doit rester valide dans le cas particulier où $V_1 = V_2 = V_3$. Cette situation peut arriver par exemple, dans un éditeur de texte dans lequel les opérations O_a et O_b suppriment le même élément du document. Dans un tel cas, l'annulation ne doit pas modifier le document.

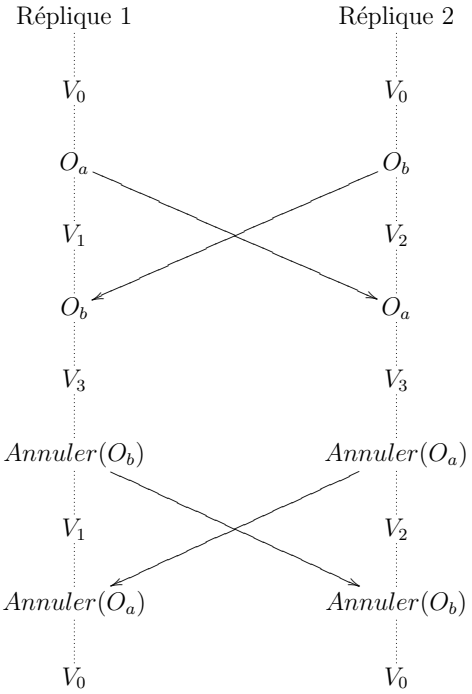


FIGURE 1.6 – Annulation de deux opérations en concurrence

1.2.3 Édition collaborative massive

L'arrivée du Web 2.0 transforme fondamentalement le Web. En effet, le contenu est désormais généré par les internautes. Ce changement implique des conséquences sur la conception d'un système d'édition collaborative : les internautes représentant 1,9 milliards d'utilisateurs [Sta10], les systèmes d'édition collaborative deviennent massifs. Par exemple, Wikipédia est un système d'édition collaborative dans lequel n'importe quel utilisateur peut contribuer, ce qui a permis de générer 16 millions d'articles en seulement 9 ans [Wik10a].

Il est donc nécessaire de trouver des algorithmes capables de gérer la charge induite par un nombre massif d'utilisateurs et de modifications. Pour cette raison, de nombreux systèmes collaboratifs se tournent vers une architecture dite pair-à-pair. Ce type de système est constitué d'ordinateurs appelés « pairs » directement interconnectés.

Androutsellis-Theotokis et Spinellis [ATS04] proposent une définition d'un système pair-à-pair.

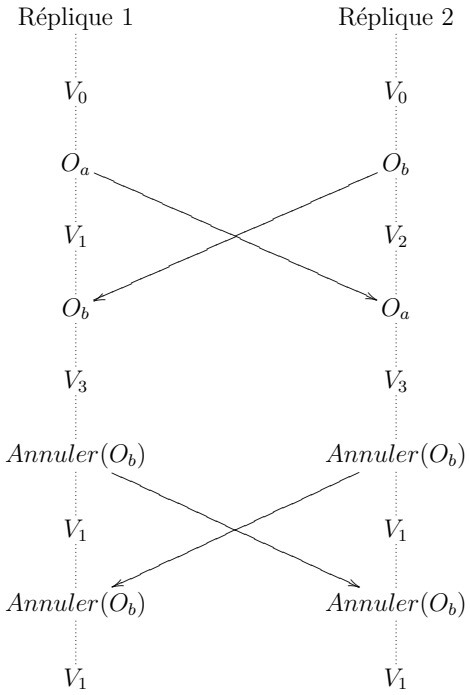


FIGURE 1.7 – Annulation d'une opération par deux répliques en concurrence

> « Les systèmes pair-à-pair sont des systèmes distribués constitués de pairs interconnectés ayant la capacité de s'auto-organiser en topologies de réseau dans le but de partager des ressources telles que du contenu, des cycles de CPU, du stockage et de la bande passante, capables de s'adapter aux pannes et d'accueillir des populations de pairs temporaires, tout en maintenant une connectivité et des performances acceptables, sans exiger l'intermédiation ou le soutien d'un serveur ou d'une autorité globale centralisée mondiale. »

Cette définition insiste sur les caractéristiques d'auto-organisation, de partage de ressources, de gestion de la dynamicité ainsi que sur le degré de décentralisation. Dans un système pair-à-pair, les données sont répliquées sur une partie des pairs du réseau. L'assignation d'une réplique à un pair peut être arbitraire [RD01] ou choisie par le pair [CRB+03]. Par conséquent, suivant les systèmes, le nombre de répliques pour une donnée peut varier de une réplique par pair à une seule réplique pour le réseau.

Les objectifs des systèmes pair-à-pair

Les systèmes pair-à-pair [ATS04] sont des alternatives aux systèmes dits client-serveur. Par rapport aux systèmes client-serveur, ils ont pour objectif d'améliorer tout ou partie des caractéristiques suivantes [DGMY03] :

Résistance à la censure

La résistance à la censure est la capacité d'un système à garantir la liberté d'expression de chacun de ses membres [PRW05]. Dans les systèmes pair-à-pair, chaque membre a le même rôle. Les réseaux pair-à-pair classiques résistent à la censure, cependant ils ne garantissent pas l'anonymat. En 2000, Ian Clarke propose Freenet [CSWH00], un système pair-à-pair garantissant l'anonymat des auteurs et des lecteurs grâce, entre autres, à l'utilisation de la décentralisation et de la cryptographie. GNUnet [FGR03] est un système de partage de fichiers garantissant l'anonymat des utilisateurs mettant en partage les données aussi bien que de ceux qui les téléchargent.

La disponibilité de service

La disponibilité témoigne du bon fonctionnement d'un service. Sa mesure est obtenu en divisant le temps durant lequel le service a fonctionné correctement par la durée totale considérée. Ainsi, la disponibilité A peut s'exprimer comme suit [Wik10b] :

$$A = \frac{T_f}{T_f + T_e}$$

avec T_f le temps de fonctionnement et T_e le temps de non-fonctionnement.

Par exemple, pour un service ayant été en panne pour une durée totale d'un jour sur un an, la disponibilité est de $A = 364/365 = 99,73\%$. Le service était, en moyenne, accessible $99,73\%$ du temps. Les systèmes pair-à-pair permettent d'accroître la disponibilité des services en utilisant les ressources des pairs dans le réseau. Par exemple, en répliquant les données sur un nombre important de pairs du réseau, on augmente en principe fortement la disponibilité de l'accès aux données.

Résistance aux pannes

La résistance aux pannes est la capacité d'un système à remplir sa fonction en cas de panne d'un ou de plusieurs de ses composants [Cri91]. Dans un système pair-à-pair, si chaque pair joue le même rôle, le non-fonctionnement d'une réplique n'entrave pas la marche du système. Les réseaux pair-à-pair sont alors considérés comme résistants aux pannes.

Support de la dynamicité

Tous les systèmes impliquant des utilisateurs doivent faire face à un problème de dynamicité de la charge. En effet, le taux d'utilisation d'un système peut être fortement variable. Par exemple, Amazon utilise seulement 10% de ses capacités en temps normal simplement pour laisser de la place à des pics occasionnels [Ama06]. Dans un système pair-à-pair, chaque utilisateur arrive et repart avec ses propres ressources. Le système s'adapte donc à la variation du nombre d'utilisateurs. On dit qu'il tolère la dynamicité.

Répartition de charge

La répartition de charge consiste a distribuer la charge de travail de façon équilibrée sur plusieurs ressources. Suivant les cas, la répartition peut se faire sur plusieurs ordinateurs, processeurs, disques durs, liens réseaux, etc. Les systèmes pair-à-pair permettent une répartition de charge par la mutualisation des ressources des pairs du réseau :

- *La capacité de stockage :* Dans un système client-serveur, le stockage est généralement à la charge du serveur. Ainsi, l'augmentation de l'espace de stockage implique un coût. Par contre, dans un système pair-à-pair, chaque pair rejoint le réseau avec sa capacité de stockage propre. Les systèmes pair-à-pair permettent de distribuer les données à stocker sur l'ensemble des pairs du réseaux.
- *La bande passante :* Afin de mettre des données à disposition, les architectures client-serveur requièrent une bande passante adaptée à la charge. Cette charge variant avec le temps, il existe un compromis entre coût et bande passante. Dans un système pair-à-pair, les échanges de données ont lieu directement entre les différents pairs du réseau. De plus, contrairement au système client-serveur, les données peuvent être récupérées depuis plusieurs sources en même temps.
- *Le processeur :* Les systèmes pair-à-pair permettent de disposer d'une puissance de calcul comparable à celle d'un supercalculateur en utilisant les processeurs de chacun des pairs. Par exemple, en mars 2010, le projet « Folding@Home » [Fol10] réunit environ 288733 processeurs ce qui constitue une puissance de calcul de 5,56 PFLOPS [1]. En comparaison, le plus puissant des supercalculateurs [Top10], « Jaguar », possède une puissance de calcul de 2,33 PFLOPS.

1.3 Conclusion

Le but de cette thèse est donc de trouver un algorithme de réplication optimiste pour les documents texte, pouvant être modifié par des opérations d'insertion, de suppression et d'annulation. Cet algorithme doit respecter le modèle de cohérence CCI communément admis pour l'édition collaborative mais aussi être compatible avec les caractéristiques des réseaux pair-à-pair ci-après :

1. **La dynamicité :** Un système passe à l'échelle en termes de dynamicité, s'il tolère de grandes variations du nombre de pairs ainsi que de sa topologie.
2. **La symétrie :** Dans un système, il existe des pairs dits « serveur », c'est à dire des pairs fournissant des fonctionnalités du système. Plus le système comporte de « serveurs », plus il est symétrique.
3. **Le nombre massif d'utilisateurs et de données :** Le système doit tolérer l'ajout d'utilisateurs ou d'objets sans subir une perte de performance notable [Neu94].

1. 10^{15} opérations en virgule flottante par seconde

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

La problématique de cette thèse est de proposer une approche pour l'édition collaborative sur un réseau pair-à-pair. Pour cela, nous devons assurer les trois critères du modèle CCI : préservation de la causalité, convergence, préservation de l'intention tout en fournissant un mécanisme d'annulation. Cette approche doit en plus être compatible avec les caractéristiques des réseaux pair-à-pair suivantes :

- la dynamicité du réseau,
- la symétrie,
- le grand nombre d'utilisateurs et de modifications.

Dans un premier temps, nous nous intéressons aux mécanismes existants permettant de respecter un des critères du modèle CCI. Puis nous présentons brièvement les solutions utilisées pour fournir une annulation de groupe. Nous discutons alors de la capacité de ces mécanismes à tolérer les caractéristiques des réseaux pair-à-pair. Finalement, nous regardons comment les approches existantes répondent à notre problématique.

2.1.1 Le modèle de cohérence CCI

Le respect de la causalité

Afin de respecter la causalité, on peut utiliser un mécanisme de diffusion atomique causale [DSU04]. Ces solutions fournissent, en plus de la causalité, une réception totalement ordonnée des messages : chaque pair reçoit les messages dans le même ordre. Afin de déterminer cet ordre, ces solutions reposent sur l'utilisation d'un estampilleur central [NCN88] ou d'un mécanisme de consensus [MR00]. Malheureusement, les solutions basées sur un *estampilleur central ne passent pas à l'échelle en termes de symétrie*. De même, les mécanismes de consensus [Lam98] ne passent pas à l'échelle en termes de dynamicité et ne passent pas à l'échelle en nombre de répliques [GBRR02].

Les solutions suivantes ne requièrent pas de réplique central. *Elles sont donc décentralisées.*

Respect de la causalité avec une propagation dite « Multicast » :

Un système distribué respecte la causalité si la réception des messages est dans un ordre causal. Certaines méthodes supposent que chaque message est à destination d'un sous-ensemble de répliques du réseau. On désigne cette propagation par « Multicast ».

Dans ce cas, certains systèmes utilisent une horloge matricielle [WB84] dont la taille est N^2 avec N le nombre de répliques dans le réseau. Il est donc nécessaire de connaître le nombre exact de répliques et leur identité, ce qui n'est pas possible dans un environnement dynamique où les répliques sont autorisées à quitter et à rejoindre le réseau à tout moment. De plus, cette matrice est envoyée avec chaque message. Par conséquent, cette solution peut poser des *problèmes de passage à l'échelle en nombre de répliques*. En effet, la taille

des messages peut devenir un obstacle lorsque le nombre de répliques devient très grand. Dans une horloge matricielle, $2N-1$ entrées sont associées à chaque réplique. L'ajout ou le retrait d'une réplique dans le système implique de modifier la matrice. Cette modification est coûteuse car elle implique l'obtention d'un consensus entre les répliques du réseau. Cette solution *ne tolère donc pas la dynamicité* des réseaux pair-à-pair.

Prakash *et al.* [PRS97] proposent une solution appelée « barrière causale » dont le coût en communication est plus faible. Chaque réplique stocke une horloge matricielle, mais envoie une matrice dont la taille est, en moyenne, C^2 avec $C \leq N$ le nombre de messages concurrents moyen. Cette solution peut améliorer grandement le passage à l'échelle en nombre de répliques. En effet, la taille de la matrice envoyée ne dépend pas du nombre de répliques dans le réseau, mais du nombre de messages générés et de la concurrence. Cette solution *permet donc un passage à l'échelle en nombre de répliques si on fait l'hypothèse que la concurrence est faible.* Elle améliore aussi la tolérance à la *dynamicité* car elle ne requiert pas l'utilisation de mécanisme de consensus en cas de variation du nombre de répliques.

Une dernière solution consiste à envoyer dans le même message, l'opération et toutes celles qui la précèdent dans l'ordre de réception [BJ87]. Cette solution est utilisée dans les mécanismes d'anti-entropie [DGH+87]. Afin de limiter le nombre d'opérations à envoyer, les répliques utilisent une horloge vectorielle pour déterminer la partie à envoyer. L'utilisation d'une horloge vectorielle [Mat89, Fid91] limite le *passage à l'échelle en termes de dynamicité.*

Respect de la causalité avec une propagation dite « Broadcast » :

De nombreuses approches font l'hypothèse que chaque message est envoyé à toutes les répliques. On appelle ce mécanisme une diffusion « broadcast ». Dans ce cas, on peut bien sûr utiliser les mécanismes précédents, mais aussi des mécanismes plus efficaces. Par exemple, Birman et al. [BSS91] proposent une réception causale avec une horloge vectorielle. Une horloge vectorielle est un tableau à une dimension contenant une entrée par réplique du réseau. Par conséquent, sa taille est proportionnelle à N. Cette solution peut poser des *problèmes de passage à l'échelle en nombre de répliques.*

Prakash *et al.* [PRS97] proposent une version des barrières causales adaptées au cas du broadcast. Le vecteur associé à chaque message a une taille comprise entre 1 et N en fonction de la concurrence dans le système.

Afin de réduire les coûts, certaines approches [Lam78, GP03] permettent une livraison causale « imparfaite ». En effet, un message peut être délivré avant ceux dont il dépend. En effet, ces solutions ne permettent pas de détecter des messages manquants. Ces approches tendent à limiter le nombre de ces violations de la causalité. L'avantage de ces solutions est le *passage à l'échelle en nombre de répliques* puisque la taille des données associées à chaque message est bornée. De plus, ces mécanismes ne requérant pas la connaissance

précise du nombre de pairs, ils tolèrent la *dynamicité*.

La convergence

Cohérence forte :

Plantikow *et al.* [PRS07] proposent un wiki pair-à-pair construit sur une table de hachage distribuée. Afin d'assurer la convergence, les auteurs proposent d'utiliser un mécanisme de consensus [Lam98] pour maintenir la cohérence entre les répliques. Ce type de mécanisme n'étant pas adapté à un grand nombre de pairs, il est appliqué uniquement sur un nombre limité de pairs responsables d'une ressource. Les mécanismes de consensus ne passent pas à l'échelle en termes de symétrie et de dynamicité. Les quorums [MR98] permettent aussi de garantir une cohérence forte, cependant ces mécanismes ont un coût incompatible avec les caractéristiques pair-à-pair [JPPMKA01].

Divergence bornée :

Afin d'améliorer les performances, certains systèmes [YV01, ABGM90] autorisant une divergence des copies définissent une métrique afin de borner la différence entre la valeur d'une copie et la valeur réelle. Par exemple [ABGM90], un utilisateur intéressé par le cours des actions d'une entreprise de produits chimiques peut être satisfait si la valeur donnée par son ordinateur est à moins de cinq pour cent du véritable cours de l'action. Il est donc inutile de mettre à jour la copie à chaque fois qu'un changement se produit. Les différentes répliques n'ont donc pas la même valeur que la réplique possédant la valeur réelle. Cette solution ne respecte pas le critère de convergence défini section 1.2.1.

La convergence à terme :

Elle assure que toutes les répliques ont la même valeur une fois le système au repos [SS05]. Les répliques peuvent donc avoir des valeurs différentes durant un temps indéfini. Ces systèmes impliquent l'existence d'un mécanisme de réconciliation permettant aux systèmes de converger vers une même valeur. Afin de réconcilier différentes répliques, plusieurs approches ont été proposées :

- *La sélection de la version :* La règle de Thomas [JT75] propose de garder la version la plus récente du document. Cette solution est notamment utilisée dans le domaine des bases de données répliquées ainsi que dans Usenet.
- *La sélection des opérations :* Seul un sous-ensemble des opérations est réellement exécuté. Pour atteindre la convergence, il est nécessaire que l'ensemble des répliques sélectionnent le même sous-ensemble d'opérations dans le même ordre. Ce type de problème est généralement résolu via l'utilisation d'un algorithme de consensus,

qui ne tolère pas la *dynamicité* ni le *nombre massif de pairs*. Cette solution est notamment utilisée dans le modèle ACF [KRSD01] et dans BAYOU [TTP+95].
- *La transformation :* Ellis *et al.* [EG89] proposent un modèle basé sur la transformation des opérations concurrentes avant de les intégrer. Cette solution est, *a priori*, compatible avec les caractéristiques pair-à-pair.
- *La commutativité :* Shapiro *et al.* [SP07] introduisent un type de données particulier dans lequel l'exécution de chaque opération concurrente est commutative. Cette solution est compatible avec les caractéristiques pair-à-pair.

Ces solutions, largement utilisées, sont détaillées dans le cadre des systèmes existants (voir section 2.2).

La préservation de l'intention

La préservation de l'intention a été introduite par Sun *et al.* [SJZ+98]. Afin de respecter les intentions, l'exécution d'une opération doit être adaptée à l'état d'exécution afin de tenir compte des opérations concurrentes. Dans le cas d'un document texte, il s'agit généralement de recalculer la position d'insertion ou de suppression [EG89, OUMI06a]. En plus de ce calcul, il peut y avoir l'insertion d'un bloc de conflit [Ber90], ou encore, l'annulation de l'effet d'autres opérations [KRSD01, JT75]. L'intention n'a jamais été formalisée et il n'existe donc pas de méthode générique pour la préserver.

2.1.2 L'annulation de groupe

L'annulation sélective globale (voir section 1.2.2) est la forme la plus générale de l'annulation de groupe. Elle permet d'annuler n'importe quelle opération. Il existe trois solutions pour fournir une annulation correcte :
- La première consiste à rejouer l'ensemble des opérations à l'exception de l'opération à annuler [CD95]. Par la suite, nous désignerons cette solution par le terme « rejeu ».
- L'annulation peut aussi s'obtenir en appliquant l'effet inverse :

  1. On applique l'inverse des opérations contenues dans l'historique en partant de la fin jusqu'à l'opération à annuler,
  2. On réapplique les opérations dans l'ordre de l'historique en commençant par celle qui suit l'opération à annuler.

  Ce mécanisme est désigné par « Undo, Skip and Redo (US&R) »[Vit84]. Malheureusement, la complexité de cette solution limite le passage à l'échelle en nombre de modifications.
- Une autre solution consiste à générer une opération d'annulation dont l'effet annule celui d'une autre opération [PK94]. On désignera cette solution sous le terme d' « annulation en avant ».

2.2 Les approches

Dans cette section, nous détaillons les approches d'édition collaborative susceptibles d'assurer tout ou partie du modèle CCI et des caractéristiques des systèmes pair-à-pair.

2.2.1 Usenet

Usenet [Sal92] est un système de forums de discussion inventé en 1979. Il est composé d'un ensemble de serveurs hébergeant les diverses discussions. Ces dernières sont répliquées sur plusieurs serveurs. Un client peut se connecter sur n'importe quel serveur pour y poster un texte. Après un certain délai, son texte sera répliqué sur tous les serveurs. En cas de mises à jour concurrentes, Usenet utilise la règle de Thomas [JT75] pour assurer la convergence.

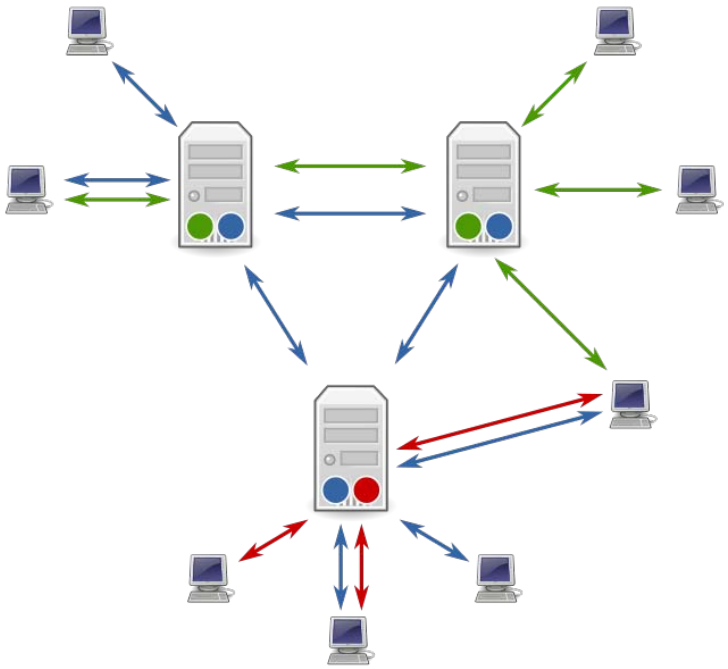


FIGURE 2.1 – Usenet : clients et serveurs (Source Wikipédia :`http://fr.wikipedia.org/wiki/usenet`)

La causalité

Les messages sont transmis périodiquement via un canal de transmission FIFO. L'ordre causal n'est pas garanti. Usenet ne respecte donc pas la causalité.

La convergence

Lors de la réception d'une version, le système choisi la version la plus récente en comparant l'horloge de la version actuelle et celle de la version reçue. La version la plus récente sera conservée. Ce mécanisme est appelé « règle de Thomas » [JT75]. Malheureusement, certaines modifications peuvent être perdues. Une fois toutes les versions reçues, toutes les répliques ont la version la plus récente, la convergence est donc assurée.

La préservation de l'intention

La préservation de l'intention définie dans la section 1.2.1 implique que toutes les modifications doivent être présentes dans le document. La perte de mise à jour implique donc une violation de cette définition. Usenet ne préserve donc pas l'intention.

L'annulation

Dans Usenet, un utilisateur peut uniquement annuler un de ses messages. Un message demandant la suppression du message est alors propagé de serveur en serveur. Ce mécanisme est une annulation de portée locale. Usenet ne fournit pas de mécanisme d'annulation sélective de portée globale.

Caractéristiques pair-à-pair

Les mécanismes utilisés par Usenet sont compatibles les caractéristiques pair-à-pair. En effet, dans la règle de Thomas, une réplique requiert uniquement l'heure de modification des messages pour assurer la convergence. Cette technique passe donc à l'échelle en nombre d'utilisateurs, en symétrie et en dynamicité. Usenet est donc compatible avec les caractéristiques pair-à-pair.

2.2.2 Les approches vérifiant uniquement la causalité

Dans cette section, nous regroupons un certain nombre de travaux du fait de leurs similitudes : ils garantissent une réception causale mais la fusion de versions concurrentes est à la charge de l'utilisateur. Il s'agit principalement des :

- systèmes de gestion de configuration distribué (DSCM) : Ils ont été introduits afin de faciliter le développement de logiciels. Ils permettent principalement la sauvegarde de l'ensemble des versions produites ainsi que l'édition collaborative. Il existe de nombreux DSCM tels que git [Tor05], darcs, mercurial, bazaar, ... Les DSCM tolèrent un très grand nombre d'utilisateurs. Ainsi, le projet « linux » implique des milliers de développeurs [Lin10] qui collaborent à l'aide de git.
- wikis pair-à-pair : Distriwiki [Mor07], DTWIKI [DB08], git-wiki [Roz08] sont des systèmes d'édition collaborative pair-à-pair utilisant la syntaxe « wiki » afin d'éditer le document.

La causalité

Dans ces systèmes, une version est définie sur les versions dont elle dépend. Un message est associé aux identifiants des messages le précédant. Ce type de propagation s'apparente à un mécanisme d'anti-entropie. Ces systèmes vérifient donc la causalité.

La convergence

Les systèmes considérés ne garantissent pas la convergence d'après la définition du modèle CCI (voir section 1.2.1). En effet, une fois les mêmes opérations reçues, deux répliques peuvent avoir des contenus différents notamment en cas de « conflit ».

Cependant, on peut considérer que le système n'est pas encore au repos. Pour tenir compte de ce cas, une définition différente de la convergence à terme a été proposée [SS05]. Après résolution du conflit, et propagation des nouvelles opérations, toutes les répliques auront la même valeur si un nouveau conflit n'apparaît pas. Ces systèmes vérifient le critère de convergence à terme tel que défini dans Saito *et al.* [SS05].

Préservation de l'intention

Le résultat de la fusion dépendant d'un utilisateur, on ne peut donc pas garantir que les relations d'ordre entre les lignes seront préservées.

L'annulation

L'annulation est intégrée dans la plupart des DSCMs ainsi que par git-wiki. Cependant, le résultat peut ne pas respecter les intentions. Les approches de wiki pair-à-pair considérées, à l'exception de git-wiki, ne fournissent pas de mécanisme d'annulation.

Caractéristiques pair-à-pair

Ces approches passent à l'échelle en nombre de répliques, en termes de symétrie et de dynamicité. Cependant, le passage à l'échelle en termes de modifications est limité par le mécanisme de propagation des modifications employées. En effet, le mécanisme d'anti-entropie a une complexité proportionnelle au nombre de modifications.

2.2.3 Le modèle ACF

L'approche ACF est un modèle générique permettant de construire des applications collaboratives. L'idée de base est de modéliser une application par des actions et des caractéristiques décrivant la sémantique de celles-ci.

Une action est similaire à une opération dans notre modèle. Les actions doivent être définies pour chaque application. Ce modèle considère trois types de caractéristiques [BBM+09] :

- Contrainte d'ordre : $A \rightarrow B$, A n'est jamais exécuté après B,
- Contrainte d'implication : $A \lhd B$ l'exécution de l'action B implique l'exécution de l'action A,
- Non commutativité : $A \nvdash B$ l'ordre d'exécution de A et B influe sur le résultat.

En combinant ces contraintes élémentaires, on peut alors exprimer d'autres contraintes :

- Atomicité : Si $A \lhd B$ et $B \lhd A$, alors, on peut exécuter soit A et B, soit aucun des deux,
- Causalité : Si $A \rightarrow B$ et $A \lhd B$, B peut être exécutée uniquement si A a réussi,
- Antagonisme : Si $A \rightarrow B$ et $B \rightarrow A$, alors, l'exécution de l'un exclut l'exécution de l'autre.

Les opérations et leurs contraintes sont stockées dans une structure de données appelée « Multi-log ». Il s'agit d'un graphe dont les nœuds sont des actions et les arêtes les contraintes entre ces actions. La contrainte d'antagonisme implique une sélection parmi les opérations du multi-log. Afin de sélectionner ces opérations, les auteurs proposent Icecube [KRSD01], un algorithme permettant de sélectionner un sous-graphe du multi-log dans lequel, toutes les contraintes sont respectées et qui contient le maximum d'actions. [2]

On peut diviser les actions du Multi-log en deux catégories :

- Les actions validées : Ces actions ont été sélectionnées par l'algorithme et ne peuvent être révoquées.
- Les actions proposées : Ces actions n'ont pas encore été validées.

L'algorithme de sélection des opérations peut fonctionner de façon :

- Centralisée [KRSD01] : Une réplique est désignée comme responsable de la sélection des sous-graphes. Une fois sélectionné, ce sous-graphe est envoyé à toutes les répliques.
- Décentralisée [SBS07] : Une partie des répliques votent pour un sous-graphe.

Babble [OS06] est un éditeur de texte basé sur le modèle ACF. Les opérations considérées sont l'insertion $Ins(p, s)$ – qui insère le texte s à la position p – et la suppression notée $Del(p, a)$ qui supprime a caractère(s) à partir de la position p. Babble utilise une translation des positions d'insertion similaire à celle du modèle des transformées opérationnelles. Une contrainte d'ordre est définie entre deux opérations dont les positions se chevauchent. Par exemple, considérons deux opérations $op_1 = Ins(0, "ade")$ et $op_2 = Ins(1, "bc")$ telles que op_2 soit exécutée après op_1. Le résultat obtenu est "abcde". Les positions d'insertion de l'opération op_2 (positions 1 et 2) sont comprises dans celles de op_1 (positions 0,1 et 2). Par définition, on a donc $op_1 \rightarrow op_2$. En comparant, les opérations locales et distantes, Babble peut générer un conflit. Pour le représenter, Babble introduit une contrainte d'exclusion mutuelle entre les opérations concernées.

2. Un poids est associé à chaque action. La sélection ne contient pas forcément le maximum d'opérations, mais elle regroupe les actions dont la somme des poids est maximum.

Causalité

La causalité est exprimée via des contraintes entre les opérations. L'algorithme de sélection a donc la charge de garantir la causalité. Cette approche préserve donc la causalité.

La convergence

Elle est obtenue via la sélection d'opérations sous réserve que les contraintes sont correctement exprimées. Par exemple, il est nécessaire de spécifier une contrainte de non-commutativité entre les actions non commutatives pour garantir la convergence. Cette approche assure la convergence si les contraintes sont correctement exprimées.

La préservation de l'intention

Dans Babble, deux insertions concurrentes à la même position sont définies comme antagonistes, c'est-à-dire, une seule insertion sera conservée. Par conséquent, un des deux éléments insérés ne sera pas dans le document. Babble n'est donc pas compatible avec la définition de la préservation de l'intention. Cependant, il est tout à possible de définir un système qui préserve l'intention telle que définie dans le modèle CCI.

Annulation

L'annulation consiste à sélectionner d'un sous-graphe ne contenant pas l'opération à annuler. Cette solution s'apparente à une annulation par rejeu. Le modèle ACF fournit un mécanisme d'annulation sélective de portée globale.

Caractéristiques pair-à-pair

Ce modèle n'est pas destiné à l'élaboration d'un système d'édition massive sur un réseau pair-à-pair. En effet, ce modèle utilise un consensus qui requiert la connaissance exacte du nombre de répliques, ce qui n'est pas compatible avec la contrainte de dynamicité des réseaux pair-à-pair. De plus, la complexité temporelle $O(n^3m^2)$ [IOM+07] dépend du nombre de répliques n dans le réseau et du nombre d'opérations m, ce qui limite le passage à l'échelle en nombre de répliques. Finalement, l'annulation a un coût proportionnel au nombre d'actions, ce qui limite le passage à l'échelle en nombre de modifications. Ce modèle n'est pas compatible avec les caractéristiques des réseaux pair-à-pair.

2.2.4 L'approche des transformées opérationnelles

Présentation :

L'approche des transformées opérationnelles [EG89] est composée de deux parties :

– Les opérations et les fonctions de transformation correspondantes : Les opérations représentent les modifications effectuées sur le document. Par exemple, pour un éditeur collaboratif, on peut définir comme opérations : $Ins(p, c)$ et $Del(p)$ avec p la position d'insertion ou de suppression dans le document et c le caractère à insérer. Les fonctions de transformation permettent de modifier une opération pour tenir compte d'opérations concurrentes.
– L'algorithme d'intégration : L'algorithme d'intégration est chargé de sélectionner les opérations à transformer.

Causalité

L'approche des transformées opérationnelles ne fait pas d'hypothèse sur le moyen d'assurer la causalité. Cependant, les solutions existantes utilisent :

– Un estampilleur central : SOCT4 [VCFS00] utilise une réplique centrale qui permet d'ordonner les opérations. Malheureusement, cette solution ne permet pas un passage à l'échelle en termes de symétrie.
– Un vecteur d'horloge [Mat89] : Un vecteur d'horloge contient une entrée par réplique, ainsi, cette solution tolère difficilement l'aspect dynamique des réseaux pair-à-pair.
– L'anti-entropie : Dans MOT2 [Mic07], les réconciliations ont lieu deux à deux, une réplique envoie la totalité de ses opérations à l'autre qui peut alors déterminer quelles opérations sont concurrentes. Malheureusement, l'envoi de la totalité des opérations posent des problèmes de charge lorsque l'édition devient massive.

Les algorithmes d'intégration existants requièrent une propriété plus forte que le respect de la causalité. En effet, ils doivent pouvoir déterminer pour chaque opération, l'ensemble des opérations concurrentes. Finalement, les approches basées sur les transformées opérationnelles assurent la causalité.

Convergence

Afin d'assurer la convergence, cette approche introduit des fonctions de transformation. Ces dernières permettent de modifier les opérations afin de tenir compte des modifications concurrentes exécutées avant. Par exemple, figure 2.2, deux répliques partagent le même document « Tortue ». La réplique 1 génère l'opération $Ins(5,\text{‘}r\text{’})$ afin d'obtenir le mot « Torture ». En parallèle, la réplique 2 rajoute un ‘s’ à la fin du mot et génère donc l'opération $Ins(6,\text{‘}s\text{’})$.

Si les opérations reçues sont intégrées telles quelles, la réplique 1 insère un ‘s’ en position 6 et obtient « Torturse » alors que la réplique 2 insère le ‘r’ en position 5 et obtient « Tortures ». Bien qu'au repos, les répliques 1 et 2 n'ont pas le même document, il y a divergence. De plus, l'intention de la réplique 2 d'insérer un ‘s’ après le ‘e’ n'est pas respectée. L'opération « ins(6, ‘s’) » ne tient pas compte du fait que l'opération concurrente « ins(5, ‘r’) » a été exécutée avant. Afin d'obtenir la convergence et le respect de l'intention, la réplique 1 doit insérer le ‘s’ en position 7. Les fonctions de transformation

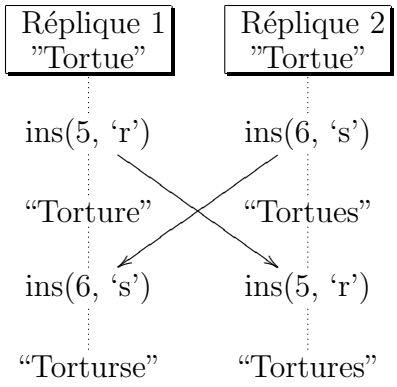


FIGURE 2.2 – Sans fonction de transformation

doivent donc transformer l'opération « ins(6, 's') » en « ins(7, 's') ». La figure 2.3 illustre le fonctionnement des fonctions de transformations.

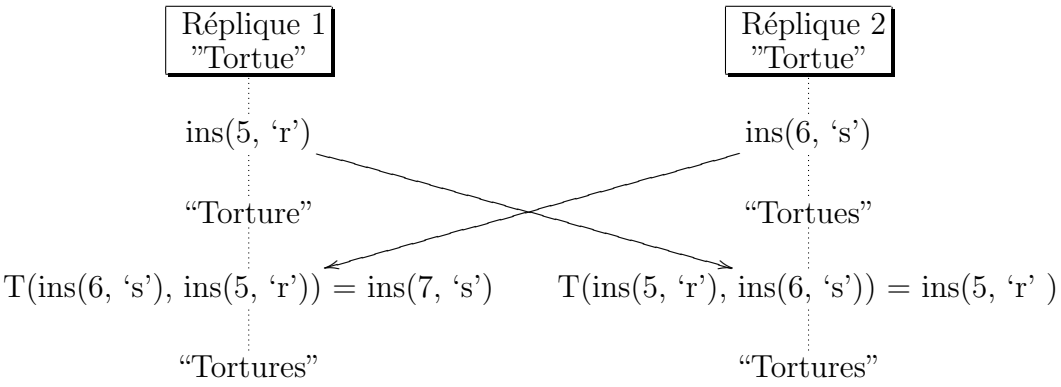


FIGURE 2.3 – Transformations

Il est donc nécessaire d'écrire des fonctions de transformation pour chaque couple d'opérations. La figure 2.4 montre un exemple de fonction de transformation.

```
1  T( Ins(p1,c1), Ins(p2, c2)):
2      if(p1 ≤ p2)
3          return Ins(p1 + 1, c1);
4      else return Ins(p1, c1);
```

FIGURE 2.4 – Exemple de fonction de transformation Insert-Insert

Afin de respecter la convergence, les fonctions de transformation doivent respecter deux propriétés [RNRG96] :

* $TP1$: La propriété $TP1$ (figure 2.5) définit une égalité entre deux états. L'état obtenu par l'exécution d'une opération op_1 sur un état S suivi par l'exécution d'une opération $T(op_2, op_1)$ doit être égal à l'état obtenu par l'exécution d'une opération

op_2 sur un état S suivi par l'exécution d'une opération $T(op_1, op_2)$:

$$TP1 : S \circ op_1 \circ T(op_2, op_1) = S \circ op_2 \circ T(op_1, op_2)$$

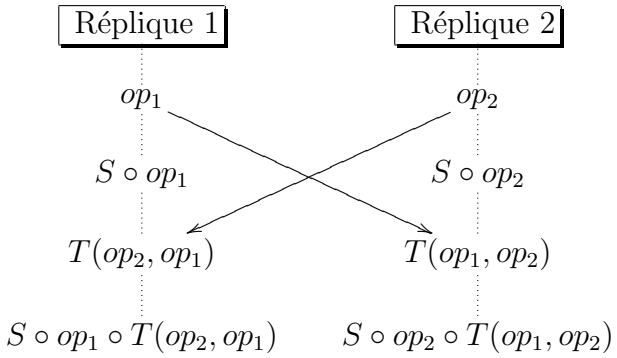


FIGURE 2.5 – Illustration de la propriété $TP1$

* $TP2$: La propriété $TP2$ (figure 2.6) assure que la transformation d'une opération par rapport à une séquence d'opérations ne dépend pas de l'ordre de transformation des opérations dans cette séquence.

$$TP2 : T(T(op_3, op_1), T(op_2, op_1)) = T(T(op_3, op_2), T(op_1, op_2))$$

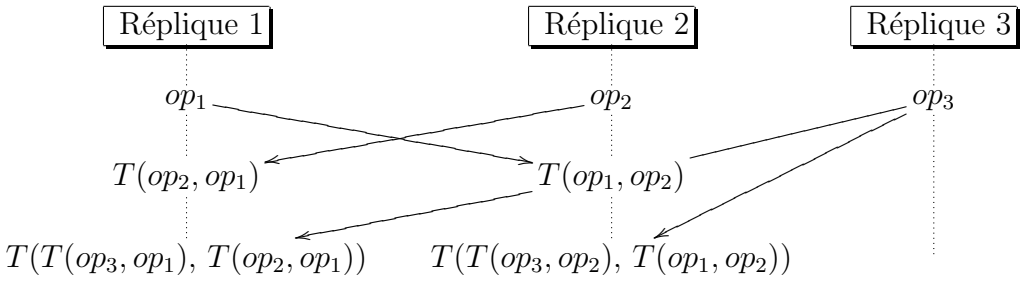


FIGURE 2.6 – Illustration de la propriété $TP2$

Malheureusement, proposer des fonctions de transformation vérifiant $TP1$ et $TP2$ est difficile [OUMI06b]. De plus, la convergence requiert n^2 transformations, n étant le nombre d'opérations concurrentes.

L'approche des transformées opérationnelles assure la convergence à condition de fournir des fonctions de transformation vérifiant les propriétés $TP1$ et $TP2$.

La préservation de l'intention

L'approche des transformées opérationnelles suppose que les fonctions de transformation vérifient l'intention. Il appartient au concepteur d'un système basé sur cette approche de s'assurer que les fonctions de transformation vérifient bien l'intention. Par exemple, les fonctions de transformation dite TTF [OUMI06b] préservent les relations d'ordre entre les éléments du document.

Annulation

Plusieurs approches d'annulation ont été proposées [PK94, RG99, Sun00, FVC04]. Certaines ne permettent pas l'annulation sélective [PK94, RG99]. D'autres [Sun00, FVC04, SS06], proposent de fournir une annulation à partir d'un système existant. Ces approches définissent trois propriétés à respecter pour obtenir une annulation correcte :

- $IP1$: L'exécution de l'inverse d'une opération immédiatement après celle-ci annule son effet :

  $$S \circ op \circ \overline{op} = S$$

  avec $\overline{op}$ l'inverse de l'opération op
- $IP2$: Cette propriété impose que la transformation d'une opération par rapport à la séquence $op \circ \overline{op}$ ne doit pas modifier l'opération.

  $$T(T(op_i, op), \overline{op}) = op_i$$

- $IP3$: Le respect de la propriété $IP3$ (figure 2.7) implique que l'effet de l'annulation d'une opération ne doit pas dépendre de sa position dans l'historique.

  $$T(\overline{op}, Seq') = \overline{T(op, Seq)}$$

  avec $Seq' = T(Seq, op)$

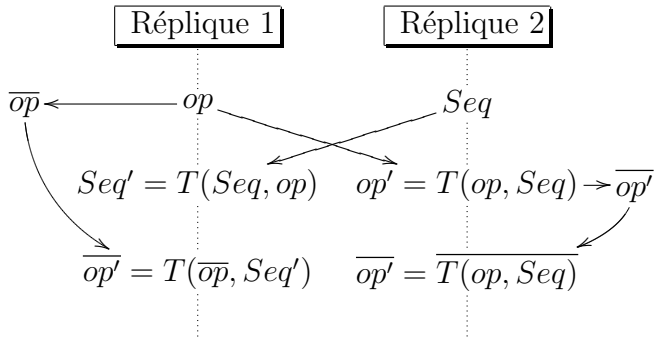


FIGURE 2.7 – Illustration de la propriété $IP3$

Dans le cas de l'annulation, les fonctions de transformation doivent donc vérifier 3 propriétés supplémentaires. Malheureusement, il n'existe aucune preuve montrant que le respect de ces propriétés est nécessaire ou suffisant pour assurer une annulation correcte. De plus, les seules fonctions de transformation vérifiant les propriétés $TP1$ et $TP2$ pour un document texte [OUMI06b], ne vérifient pas la propriété $IP2$.

Nous avons proposé une solution d'annulation appelée « Compensation » [WUM08]. Contrairement aux approches existantes, l'annulation est obtenue directement par les fonctions de transformation. Cette solution permet d'obtenir l'annulation quel que soit l'algorithme d'intégration. Par contre, la compensation ne respecte pas nécessairement la définition de l'annulation définie dans la section 1.2.2.

Elle est composée des étapes suivantes :

1. Définition des opérations de compensation : Pour chaque action du système, il est nécessaire de définir une opération de compensation. Cette opération peut ne pas appartenir à l'ensemble des opérations initiales.
2. Écriture des fonctions de transformation pour les opérations de compensation.
3. Vérification des propriétés $TP1$-$IP3$ ou $TP1$-$TP2$-$IP3$ suivant l'algorithme d'intégration.

Contrairement aux approches existantes, la compensation requiert uniquement la propriété $IP3$ en plus des propriétés requises par l'algorithme d'intégration.

Sun *et al.* [SS06, SS09] proposent COT, un nouvel algorithme d'intégration intégrant un mécanisme d'annulation spécifique. Ce dernier introduit la notion de « vecteur de contexte » : une horloge vectorielle à laquelle se rajoute un ensemble d'opérations. Malheureusement, l'ensemble contenu dans un vecteur de contexte a une taille non bornée.

Cette nouvelle structure permet notamment de définir chaque opération d'annulation sur l'état de génération de l'opération à annuler. Par exemple, supposons un système dans lequel deux opérations op_1 et op_2 ont été générées dans cet ordre et intégrées sur l'ensemble des répliques. Chaque réplique est dans un état $S \circ op_1 \circ op_2$. Si une réplique décide d'annuler op_1, elle génère une opération notée $\overline{op_1}$ qui sera définie, grâce au vecteur de contexte, sur l'état $S \circ op_1$. Malheureusement, la relation de causalité entre les opérations $\overline{op_1}$ et op_2 n'est pas capturée. Le vecteur de contexte ne permet donc pas une réception causale.

Caractéristiques pair-à-pair

La transformation d'opérations permet de garantir la convergence. Cette technique ne limite pas le passage à l'échelle. Cependant, elle requiert une détection de la concurrence entre toutes les opérations. Malheureusement, les mécanismes permettant une telle détection ne passent pas à l'échelle en nombre de répliques ou en nombre d'opérations. L'approche des transformées opérationnelles ne sont pas compatibles avec les caractéristiques des réseaux pair-à-pair.

2.2.5 Le modèle « type de données répliqué commutatif » (CRDT)

Un type de données CRDT [SP07] est un type de données pour lequel l'exécution d'opérations concurrentes est commutative. On peut facilement démontrer qu'un système basé sur un type de données CRDT respectant la causalité garantit la convergence [SP07]. En effet, d'après la condition de précédence, seules les opérations ayant une relation de causalité ne commutent pas. Or, ces opérations sont nécessairement exécutées dans le même ordre sur toutes les répliques. Par conséquent, si le système assure la causalité, le type de données CRDT assure la convergence.

Les types de données abstraits répliqués

Les types de données abstraits répliqués [RKL06] sont basés sur la commutativité des opérations concurrentes. Les auteurs proposent, comme structure de données répliquée, un tableau sur lequel deux actions sont possibles :

- Écriture : une réplique peut affecter une valeur à une case du tableau.
- Ajout en file : une réplique peut ajouter une nouvelle valeur à la fin du tableau.

Chaque opération est envoyée avec un vecteur d'horloge [Mat89]. Ce dernier est utilisé pour garantir une réception causale, mais aussi pour assurer la convergence en cas de concurrence. En effet, les auteurs utilisent un ordre total obtenu à partir du vecteur d'horloge [SJZ+98].

- Si plusieurs répliques affectent des valeurs différentes en concurrence, une valeur sera choisie en utilisant le vecteur d'horloge associé à chaque opération.
- En cas d'ajouts concurrents en file, l'ordre des éléments ajoutés est déterminé en utilisant le vecteur d'horloge.

Cependant, une telle structure de données ne peut être utilisée pour construire un éditeur de texte collaboratif. En effet, il n'est pas possible de supprimer un élément, ni d'en insérer un ailleurs qu'en fin du tableau.

Woot

Dans Woot, un identifiant unique est associé à chaque élément du document. Ces identifiants sont partiellement ordonnés, en effet, chaque identifiant est envoyé avec son précédent et son suivant. Ainsi un document Woot est un diagramme de Hasse (voir figure 2.8).

L'algorithme Woot [OUMI06a] calcule alors une extension linéaire de ce diagramme. Par exemple, pour le diagramme décrit figure 2.8, on obtient « L_B, L_3, L_2, L_1, L_E ».

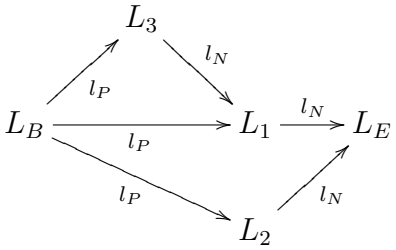


FIGURE 2.8 – Exemple de digramme pour un document Woot

Malheureusement, il n'est pas possible de supprimer un élément. En effet, il n'est plus possible d'intégrer une insertion si l'élément précédent ou l'élément suivant ne sont plus dans le document. Par conséquent, pour supprimer un élément, on le remplace par une pierre tombale possédant le même identifiant. La taille du document ne peut donc que grossir, ce qui limite le passage à l'échelle en nombre de modifications.

TreeDoc

TreeDoc [SP07, PMSL09] utilise un arbre binaire pour représenter le document. Le

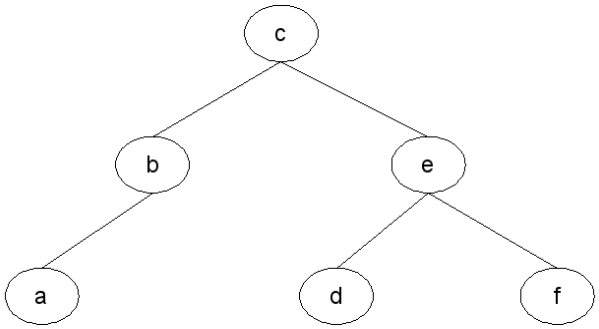


FIGURE 2.9 – TreeDoc

contenu est représenté par un parcours infixe de l'arbre. Par exemple, sur la figure 2.9, le document contient « abcdef ». Un document TreeDoc peut être modifié de deux façons :

- $Insertion(chemin, contenu)$: *chemin* est le chemin du nœud à insérer dans l'arbre binaire stocké sous forme de liste binaire, et *contenu* le contenu du nœud à insérer.
- $Suppression(chemin)$: *chemin* est le chemin du nœud à supprimer.

Un nœud peut toujours être inséré entre deux autres consécutifs. L'insertion se fait en l'ajoutant comme fils droit du nœud précédent ou comme fils gauche du nœud suivant. Figure 2.10, pour insérer un « X » entre « d » et « e », on ajoute un nœud contenant « X » comme fils droit du nœud contenant « d ». Pour chaque insertion, une opération $Insertion(chemin, contenu)$ est envoyée à toutes les répliques pour assurer la convergence. Chaque nœud supprimé est remplacé par une pierre tombale s'il possède au moins un fils. Chaque suppression est représentée par une opération $Suppression(chemin)$ envoyée à toutes les répliques.

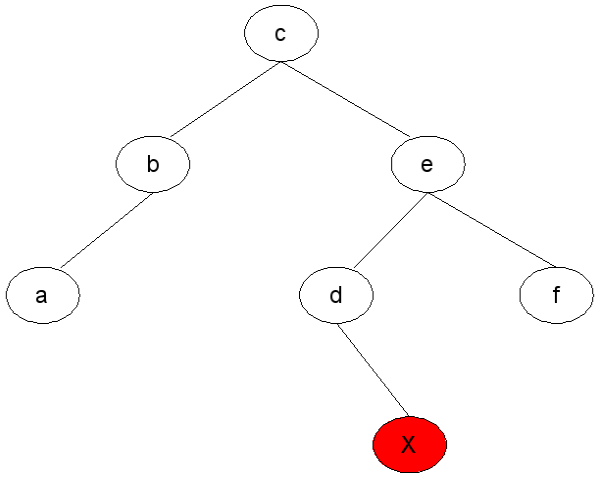


FIGURE 2.10 – TreeDoc : Insertion

Dans ce système, plusieurs utilisateurs pouvent insérer des nœuds à la même position dans l'arbre. Afin de tenir compte de ce cas, les auteurs introduisent la notion de « super-nœud ». Un super-nœud contient plusieurs nœuds ordonnés par un identifiant.

Figure 2.11, une réplique insère un « Y » entre « d » et « e » en même temps qu'une seconde réplique insère un « X » à la même position. Le résultat présenté suppose que l'identifiant du nœud contenant « Y » est inférieur à celui du nœud contenant « X ».

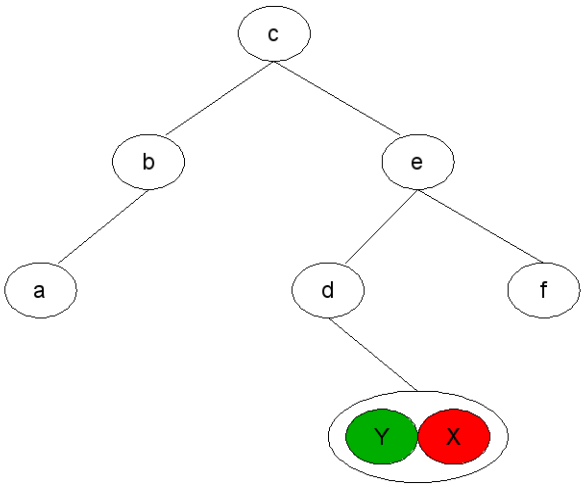


FIGURE 2.11 – TreeDoc : Super-Nœud

L'introduction de super-nœud modifie les chemins dans l'arbre. En effet, le dernier identifiant du chemin doit contenir l'identifiant. De plus, une réplique peut insérer entre « Y » et « X » dans l'exemple précédent. Dans ce cas, il est alors nécessaire de spécifier le nœud auquel il est rattaché, et donc, il est nécessaire de préciser l'identifiant du nœud père.

Au cours de l'édition, l'arbre peut devenir déséquilibré ou contenir un grand nombre de pierres tombales ce qui provoque une baisse de performance. Afin de résoudre ce problème, les auteurs proposent deux opérations dites structurelles. L'opération « flatten » consiste à stocker une partie de l'arbre, sans les pierres tombales, sous une forme canonique. Une autre opération appelée « explode » permet de re-transformer la forme canonique sous forme d'un arbre. Ces dernières ne commutant pas avec les opérations d'éditions, un consensus entre les répliques est requis.

La causalité

Dans Roh *et al.*, la causalité est respectée par l'utilisation de vecteurs d'horloge. Dans Woot et TreeDoc, les auteurs ne proposent pas de solution permettant de vérifier la causalité. Cependant, ces approches peuvent être utilisées avec n'importe quel mécanisme permettant une diffusion causale telle que les barrières causales [PRS97], qui est compatible avec les caractéristiques pair-à-pair.

La convergence

Dans les CRDTs, la convergence est obtenue par la commutativité de toutes les opérations concurrentes. Les structures de données abstraites utilisent une horloge vectorielle, alors que WOOT et TreeDoc utilisent des pierres tombales pour assurer la commutativité. Les CRDTs assurent naturellement la convergence grâce à la commutativité.

L'intention

Dans les CRDTs existants, l'intention est préservée en spécifiant un ordre partiel entre les éléments. Cependant, les CRDTs n'imposent pas de méthode particulière pour assurer les intentions. Les CRDTs existants préservent l'intention.

L'annulation

Le problème de l'annulation a été traité uniquement pour WOOT [RWSM+09]. Cette solution propose une annulation en avant et ne limite pas le passage à l'échelle de l'approche. Cependant, ce mécanisme n'est pas générique et ne peut pas être adapté à toutes les approches CRDT. Il n'existe pas de mécanisme générique pour fournir une annulation pour les CRDTs.

Caractéristiques pair-à-pair

Le type de données CRDT est théoriquement compatible avec les caractéristiques pair-à-pair. Cependant, les approches existantes obtiennent la commutativité via des horloges vectorielles ou des pierres tombales. Ces techniques limitent la compatibilité avec les caractéristiques pair-à-pair. En effet, les horloges vectorielles requièrent la connaissance exacte du nombre de répliques à tout moment, ce qui n'est pas compatible avec la dynamicité.

Les pierres tombales sont stockées dans le document et interviennent donc dans la complexité des algorithmes. Malheureusement, dans WOOT, le nombre de pierres tombales ne peut que grossir, limitant le passage à l'échelle en nombre d'éditions. Concernant TreeDoc, une partie des pierres tombales peut être purgée. Le nombre de pierres tombales restants dépend des positions d'insertion, dans le pire cas, $n - 1$ nœuds parmi n sont des pierres tombales. En pratique, le nombre de pierres tombales est certainement bien moins important. Afin de réduire le nombre de pierres tombales, des techniques de purge [JT75, SJZ+98] peuvent être utilisées. Malheureusement, ces dernières ne sont pas compatibles avec les caractéristiques des réseaux pair-à-pair.

2.3 Conclusion

Dans un premier temps, nous avons brièvement présenté les approches permettant de vérifier une partie du modèle de cohérence ou l'annulation. Nous avons alors discuté de

leur compatibilité avec les caractéristiques pair-à-pair. Ensuite, nous avons présenté différentes approches d'édition collaborative susceptibles de résoudre le problème de l'édition collaborative massive sur réseau pair-à-pair. Le tableau ci-dessous synthétise le niveau de cohérence de chaque approche suivant les critères que nous avons présentés dans les sections 1.2.1 et 1.3.

Approche	Caractéristiques collaborative				Caractéristiques pair-à-pair		
	Cohérence			Annulation	Dynamicité	Symétrie	Édition massive
USENET	-	C	-	-	X	X	X
DSCM/P2PWiki	C	-	-	X	X	X	X
Babble (ACF)	C	C	-	X	-	-	-
SOCT4 (OT)	C	C	I	X	-	-	-
SOCT2 (OT)	C	C	I	X	-	X	-
MOT2 (OT)	C	C	I	X	X	X	-
[RKL06] (CRDT)	C	C	-	-	-	-	X
WOOT (CRDT)	-	C	I	X	X	X	-
TreeDoc (CRDT)	C	C	I	-	X	X	X

Niveau de cohérence :
Causalité - Converge - Intention

Finalement, il existe plusieurs approches respectant les caractéristiques pair-à-pair telles que TreeDoc, Usenet ou les gestionnaires de configuration distribués mais qui ne vérifient pas l'ensemble du modèle de cohérence avec un mécanisme d'annulation. D'un autre côté, certaines approches fournissent une annulation et respectent le modèle CCI mais ne sont pas compatibles avec les caractéristiques pair-à-pair. Notre problème consiste à trouver une solution vérifiant le modèle de cohérence CCI avec l'annulation, tout en étant compatible avec les caractéristiques des réseaux pair-à-pair.

Chapitre 3

Modèle pour l'édition collaborative

Sommaire

3.1 Introduction

Les systèmes d'édition collaborative existants cherchent à assurer le modèle de cohérence CCI (préservation de la causalité, convergence et préservation de l'intention). Malheureusement, ce modèle est formalisé et interprété de plusieurs façons différentes. Par exemple, nous avons vu dans le chapitre 1, différentes interprétations de la propriété : « préservation de la causalité », ainsi que l'absence de formalisme pour la préservation de l'intention. Afin de lever toute ambiguïté lors de la lecture des chapitres suivants, nous proposons une formalisation d'un système d'édition collaborative, ainsi que du modèle de cohérence CCI pour les systèmes d'édition collaborative de texte.

3.2 Modèle de système d'édition collaboratif

Dans un premier temps, nous définissons un modèle générique vis-à-vis du type de données édité. Ensuite, nous nous intéressons plus particulièrement aux documents texte.

3.2.1 Modèle général

Notre modèle de réplication est basé sur la réplication optimiste [SS05]. Dans les systèmes de réplication optimiste, les mêmes données sont stockées sur plusieurs répliques. Sans perte de généralité, nous considérons que chaque réplique héberge une copie d'un document.

Définition 1 (Ensemble des répliques)**.** *On note* $\mathbb{R}$ *l'ensemble des répliques d'un système d'édition collaborative.*

En pratique, une réplique ne connaît généralement pas l'ensemble des répliques du système. Dans le cas d'un système pair-à-pair, chaque réplique du système possède le même rôle. Afin de différencier les répliques d'un document, un identifiant unique est associé à chaque réplique. Cet identifiant est utilisé dans notre modèle et n'existe pas nécessairement dans les systèmes de réplication optimiste.

Un utilisateur peut modifier une partie du document partagé à tout moment sans consulter au préalable les autres répliques. On appelle « valeur » le contenu du document partagé. On suppose que, sur une réplique donnée, les modifications sont effectuées séquentiellement. À un moment donnée, les répliques peuvent donc avoir des valeurs différentes.

Dans notre modèle, une modification effectuée sur une réplique est représentée par une *opération.*

Définition 2 (Ensemble des opérations)**.** *On note* $\mathbb{H}$ *l'ensemble des opérations d'un système d'édition collaborative.*

Une opération est dite « locale » pour une réplique si elle a été générée sur cette réplique. Similairement, une opération reçue provenant d'une autre réplique est dite « distante ». Une opération est donc locale pour une réplique et distante pour toutes les autres répliques.

On considère qu'une réplique change d'état immédiatement à chaque réception d'une opération. Un état est caractérisé par l'identifiant de la réplique et par l'ensemble des opérations reçues et par l'ordre dans lequel la réplique les a reçues.

Définition 3 (État d'une réplique)**.** *L'état S d'une réplique est défini par un triplet $(H_S, \rightarrow_S, R)$ avec $R \in \mathbb{R}$ l'identifiant de la réplique, $H_S \subseteq \mathbb{H}$ un historique et $\rightarrow_S \subseteq \mathbb{H} \times \mathbb{H}$ une relation d'ordre strict total sur H_S.*

Cette définition garantit l'unicité de chaque état. En effet, même si deux répliques ont reçu les mêmes opérations dans le même ordre, nous faisons la distinction entre ces états via l'identifiant des répliques. Un nouvel état peut être créé par la livraison d'une opération locale ou distante. Cette opération est alors ajoutée à la « fin » de l'historique.

Définition 4 (Exécution d'une opération)**.** *On note $S \circ op$ l'état obtenu par l exécution d'une opération op, locale ou distante, sur une réplique se trouvant dans l'état S. Soit $S = (H_S, \rightarrow_S, R)$, on a $S \circ op = (H_S \cup \{op\}, \rightarrow_S \cup \{(op_x, op) \mid op_x \in H_S\}, R)$. Dans le cas d'une opération locale, on note $St(op)$ l'état de génération de $op \in \mathbb{H}$ sur une réplique.*

Dans un système réel, il n'existe pas forcément une fonction calculable permettant de retrouver l'état de génération d'une opération. Cependant, l'état $St(op)$ a nécessairement existé. Dans notre modèle, on identifie cet état par $St(op)$. Par exemple, $St(op)$ peut être identifié par une horloge vectorielle [Mat89].

Dans un système d'édition collaborative donné, seul un sous-ensemble des états peut être atteint. En effet, un état est défini par un ensemble d'opérations ordonnées, or certains ordres peuvent ne pas être permis par le système.

Par exemple, considérons un système d'édition collaborative dans lequel une opération $Create(e, S)$ crée un élément e sur un état S et une autre $Delete(e, S')$ le supprime sur un état S'. Il est possible que ce système ne permette pas de générer ou de recevoir la seconde opération si la première n'appartient pas à l'historique.

Nous définissons donc un prédicat pour modéliser d'éventuelles préconditions à la génération d'une opération.

Définition 5 (Génération d'une opération)**.** *On note $Generable(op, S)$, le prédicat représentant la possibilité de générer ou non une opération $op \in \mathbb{H}$ sur un état S.*

Ce prédicat est vrai si l'opération op est générable sur un état S. Pour l'exemple précédent, on peut alors définir la condition de génération d'une suppression :

$$\forall S = (H_S, \rightarrow_S, R), Generable(Delete(e, S), S) = (\exists S'.Create(e, S') \in H_S) \wedge (\nexists S'.Delete(e, S') \in H_S)$$

Similairement, une opération peut ne pas être livrée dans un certain état. Reprenons l'exemple précédent dans le lequel, une réplique crée un élément avec une opération $Create(e, S)$ puis le supprime avec une opération $Delete(e, S)$. Si le système ne permet pas la génération de la suppression avant celle la création, on peut imaginer qu'une réplique distante ne doit pas exécuter l'opération $Delete(e, S)$ de suppression avant d'avoir exécuté celle de création.

Nous définissons donc un prédicat pour modéliser d'éventuelles préconditions à la livraison d'une opération.

Définition 6 (Condition de livraison d'une opération)**.** *On note $Livrable(op, S)$, le prédicat qui spécifie si une opération op peut être livrée par la réplique dans l'état S.*

Pour l'exemple précédent, on peut alors définir la condition de livraison d'une suppression :

$$\forall S = (H_S, \rightarrow_S, R), Livrable(Supp(e, S'), S) = \exists S''.Create(e, S'') \in H_S \wedge Delete(e, S') \notin H_S)$$

Ce prédicat permet aussi de modéliser par exemple, une réception des opérations totalement ordonnées [DSU04] ou une réception causale [KS98]. On peut désormais définir l'ensemble des états atteignables pour un système.

Définition 7 (L'ensemble des états atteignables)**.** *On appelle $\mathbb{S}$ l'ensemble des états atteignables du système. Par définition, les états initiaux des répliques appartiennent à $\mathbb{S}$.*

$$\forall n \in \mathbb{R}.(\{\}, \{\}, n) \in \mathbb{S}$$

Un nouvel état peut être créé par l'exécution d'une opération locale.

$$\forall S = (H_S, \rightarrow_S, n) \in \mathbb{S}.\forall op \in \mathbb{H}.op \notin H_S \wedge Generable(op, S) \Rightarrow S \circ op \in \mathbb{S}$$

Un réplique change aussi d'état lors de l'exécution d'une opération distante.

$$\forall S = (H_S, \rightarrow_S, n) \in \mathbb{S}.\forall op \in \mathbb{H}.St(op) \circ op \in \mathbb{S} \wedge op \notin H_S \wedge Livrable(op, S) \Rightarrow S \circ op \in \mathbb{S}$$

À chaque état d'une réplique, il correspond une valeur du document. Cette dernière peut être calculée à partir des opérations exécutées et de l'ordre d'exécution de ces opérations.

Définition 8 (Document collaboratif)**.** *On note $\mathbb{D}$, l'ensemble des valeurs d'un document collaboratif. On appelle $D : \mathbb{S} \mapsto \mathbb{D}$ la fonction qui retourne la valeur du document de la réplique dans l'état $S \in \mathbb{S}$.*

L'ensemble des valeurs d'un document $\mathbb{D}$ dépend du type de document édité. Par exemple, pour un document texte, $\mathbb{D}$ contient l'ensemble des textes possibles.

Un état S contient l'ensemble des opérations livrées et l'ordre de livraison. On suppose donc, l'existence d'un mécanisme capable de calculer la valeur du document à partir de ces informations.

Afin de rendre notre modèle le plus générique possible, nous considérons l'annulation sélective [PK94] avec une portée globale. En effet, à partir de cette fonctionnalité, il est possible de fournir toutes les stratégies d'annulation [Vid02]. Nous définissons une opération *Annuler* pour représenter l'annulation d'une opération.

Définition 9 (Fonction d'annulation)**.** *Nous supposons l'existence d'une fonction Annuler* : $\mathbb{H} \times \mathbb{S} \mapsto \mathbb{H}$ *qui représente l'annulation d'une opération effectuée par une réplique dans un état donné.*

Par exemple, l'opération $Annuler(op, S)$ représente une annulation de l'opération op effectuée sur l'état S. Cette fonction permet donc de différencier deux annulations de la même opération effectuées à sur des répliques différentes ou dans des états différents. Comme les opérations classiques, une opération d'annulation est unique. Dans notre modèle, cette opération est traitée comme toute autre opération, c.-à-d., elle est envoyée à toutes les répliques qui la ré-exécuteront.

Finalement, nous pouvons définir un système d'édition collaborative.

Définition 10 (Instance de système d'édition collaborative)**.** *Une instance d'un système d'édition collaborative sans annulation est définie par le tuple :*

$$\langle \mathbb{D}, \mathbb{H}, D, Livrable, Generable \rangle$$

De même, une instance d'un système d'édition collaborative avec annulation est définie par

$$\langle \mathbb{D}, \mathbb{H}, D, Livrable, Generable, Annuler \rangle$$

3.2.2 Modèle pour l'édition collaborative de documents texte

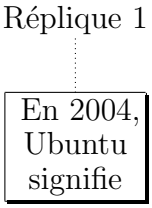
Les définitions précédentes sont génériques et s'appliquent à tous les systèmes d'édition collaborative basés sur une réplication optimiste. Dans la suite de cette section, les définitions sont spécifiques aux systèmes d'édition collaborative de document texte.

Dans le cas de l'édition de document texte, on considère le document comme un ensemble totalement ordonné d'éléments identifiés de manière unique. Cette identification des éléments n'est pas forcément explicite pour toutes les approches, cependant, on suppose qu'elle existe. Par exemple, pour l'approche des transformées opérationnelles, un identifiant unique peut être constitué du vecteur d'horloge caractérisant un état et de la position de l'élément dans cet état.

Ces éléments peuvent représenter des caractères [EG89], des lignes [Ber90], des paragraphes, mais aussi, des images, des vidéos, etc. Nous considérons chaque élément comme unique. Par exemple, pour des caractères, notre modèle différencie les deux 't' du mot « bicyclette ».

Définition 11 (Ensemble des éléments d'un système d'édition collaborative de texte)**.** *On note $\mathbb{E}$ l'ensemble des éléments d'un système d'édition collaborative de texte. Un document texte est un ensemble ordonné d'éléments. $D(S) = (E_S, \prec_S)$ définit la valeur du document texte d'une réplique dans l'état S avec $E_S \subseteq \mathbb{E}$ et $\prec_S \subset \mathbb{E} \times \mathbb{E}$ une relation d'ordre total strict défini sur E_S. L'ensemble des valeurs d'un document texte est alors noté $\mathbb{D} = \{(E_S, \prec_S) \mid S \in \mathbb{S}\}$*

Par exemple, le texte suivant :



contient les éléments « En 2004, », « Ubuntu » et « signifie » avec

$$\text{« En 2004, »} \prec_S \text{« Ubuntu »} \prec_S \text{« signifie »}$$

Remarque : Dans la plupart des éditeurs collaboratifs de texte [Ber90, OUMI06a, OUMI06b, Tor05], les documents texte peuvent être modifiés par deux types d'opérations :

1. Insertion d'un élément.
2. Suppression d'un élément.

La mise à jour d'un élément est donc traduite par la suppression de l'ancien élément et l'insertion d'un nouveau.

Par conséquent, dans notre modèle, nous considérons les opérations suivantes :

- l'insertion notée $Ins(e, S)$, avec $e \in \mathbb{E}$ l'élément à insérer et $S \in \mathbb{S}$ l'état de génération de cette opération,
- la suppression notée $Supp(e, S)$, avec $e \in \mathbb{E}$ l'élément à supprimer et $S \in \mathbb{S}$ l'état de génération de cette opération.

Finalement, dans notre modèle d'édition collaborative de texte :

$$\mathbb{H} = \{Ins(e, S) \mid e \in \mathbb{E} \wedge S \in \mathbb{S}\} \cup \{Supp(e, S) \mid e \in \mathbb{E} \wedge S \in \mathbb{S}\}$$

Chaque élément e est unique et contient sa position (selon $\prec_S$) dans le document. Par exemple, on peut définir un élément $e \in \mathbb{E}$ par $e = \langle position, contenu, S\rangle$ avec *position* la position de l'élément dans le document lors de son insertion, *contenu* le contenu de l'élément et S l'état de génération de l'élément. Dans ce cas, la position dans le document est relative à S et on suppose l'existence d'un mécanisme permettant calculer la position de cet élément sur les autres états. Par exemple, l'élément $\langle 3, \text{« } \dots \text{ »}, S_1\rangle$ se situe à la position 3 dans l'état S_1 et, à la position 4 dans l'état $S_1 \circ Ins(2, e)$. Par la suite, nous

utiliserons cette solution dans les exemples. Cependant, il existe d'autres façons de définir un élément, on peut aussi, définir sa position par l'ensemble des éléments le précédant et l'ensemble des éléments le suivant.

La Figure 3.1 illustre la modification d'un document via une opération d'insertion. Le document dans l'état S_{10} est modifié par un utilisateur qui génère une opération $ins(\langle 0, \text{“Ubuntu”}, S_{10}\rangle, S_{10})$. L'exécution de cette opération génère une nouvelle version du document. Le document est dans l'état S_{11}. L'exécution des opérations $ins(\langle 0, \text{“En 2004,”}, S_{11}\rangle, S_{11})$ et $del(\langle 0, \text{“En 2004,”}, S_{11}\rangle, S_{12})$ crée deux nouveaux états S_{12} et S_{13}.

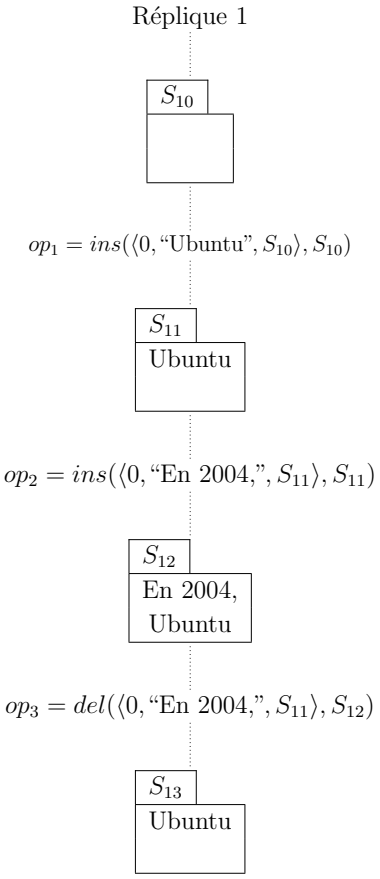


FIGURE 3.1 – Génération

Ainsi dans cet exemple, l'état généré par $ins(\langle 0, \text{“En 2004,”}, S_{11}\rangle, S_{11})$ est S_{12}. Par définition on a $St(Ins(e, S)) = S$. Contrairement à S, qui identifie de façon unique un état d'une réplique, $D(S)$ représente le contenu de cette réplique et n'est donc pas unique. Par exemple, Figure 3.1, on a $D(S_{11}) = D(S_{13})$.

3.3 Modèle de cohérence

La cohérence dans un système d'édition collaborative est orientée utilisateur. Sun et al. [SJZ+98] définissent un système d'édition collaborative comme correct s'il respecte les

conditions suivantes : préservation de la causalité, de la convergence et de l'intention.

La causalité est un critère générique défini pour tout système d'édition collaborative, y compris pour les éditeurs de texte. Par contre, la convergence et la préservation de l'intention sont des critères spécifiques à chaque instance. La définition proposée dans cette section ne s'applique donc qu'aux systèmes d'édition collaborative de texte.

3.3.1 Respect de la causalité

Dans le modèle de cohérence CCI, la causalité est définie comme une « relation de précédence » [Lam78]. Il s'agit d'une relation d'ordre partiel, notée $\rightarrow$, sur l'ensemble des opérations $\mathbb{H}$.

Dans notre modèle, cette relation de précédence se définit par :

Définition 12 (Relation de précédence causale)**.** *Soit op_1 et op_2 deux opérations de $\mathbb{H}$ telles que $(H_S, \rightarrow_S, n) = St(op_2)$ avec $St(op_2) \in \mathbb{S}$. Nous avons $op_1 \rightarrow op_2$ si et seulement si $op_1 \in H_S$.*

On suppose aussi l'existence d'une relation $Sem(op_1, op_2)$ représentant les dépendances dites sémantiques pour un système d'édition collaborative.

Définition 13 (Dépendance sémantique)**.** *Soit $Sem(op_1, op_2)$ un prédicat vrai si l'opération op_2 dépend sémantiquement de l'opération op_1. On a alors $Sem(op_1, op_2) \Rightarrow op_1 \rightarrow op_2$.*

Par exemple, dans un système d'édition collaborative de texte, on peut définir qu'une opération de suppression d'un élément dépend sémantiquement de l'opération d'insertion de cet élément : $Sem(Ins(e, S), Supp(e, S'))$. La précédence causale inclut la causalité « sémantique », en effet, les causes de la génération de *op* sont forcément antérieures à sa génération.

Définition 14 (Livraison conforme à l'ordre causal)**.**
Un système collaboratif assure une livraison causale si et seulement si :

$$\forall S = (H_S, \rightarrow_S, R_S) \in \mathbb{S}, ((H_S \times H_S) \cap \rightarrow) \subseteq \rightarrow_S$$

Afin d'illustrer notre formalisation de la relation de précédence, nous considérons l'exemple Figure 3.2. Deux répliques ont le même document. Initialement, on a $H_{S_{11}} = H_{S_{21}} = \{op_1\}$ avec $op_1 = ins(\langle 0, \text{“Ubuntu”}, S_{10}\rangle, S_{10})$. Dans ce document, l'utilisateur 1 insère une ligne « En 2004, » en créant une opération $op_2 = ins(\langle 0, \text{“En 2004,”}, S_{11}\rangle, S_{11})$. L'état devient alors $(H_{S_{12}} = \{op_1, op_2\}, \rightarrow_{S_{12}}, R_1)$ avec $\rightarrow_{S_{12}} = \{(op_1, op_2)\}$. Puis le premier utilisateur supprime la ligne « En 2004, » à l'aide de l'opération $op_3 = del(\langle 0, \text{“En 2004,”}, S_{11}\rangle, S_{12})$. L'état de génération de cette nouvelle opération est $S_{13} = (H_{S_{13}} = \{op_1, op_2, op_3\}, \rightarrow_{S_{13}}, R_1)$ avec $\rightarrow_{S_{13}} = \{(op_1, op_2), (op_2, op_3), (op_1, op_3)\}$. On a $op_2 \in H_{S_{13}}$ ce qui implique, par définition, $op_2 \rightarrow op_3$.

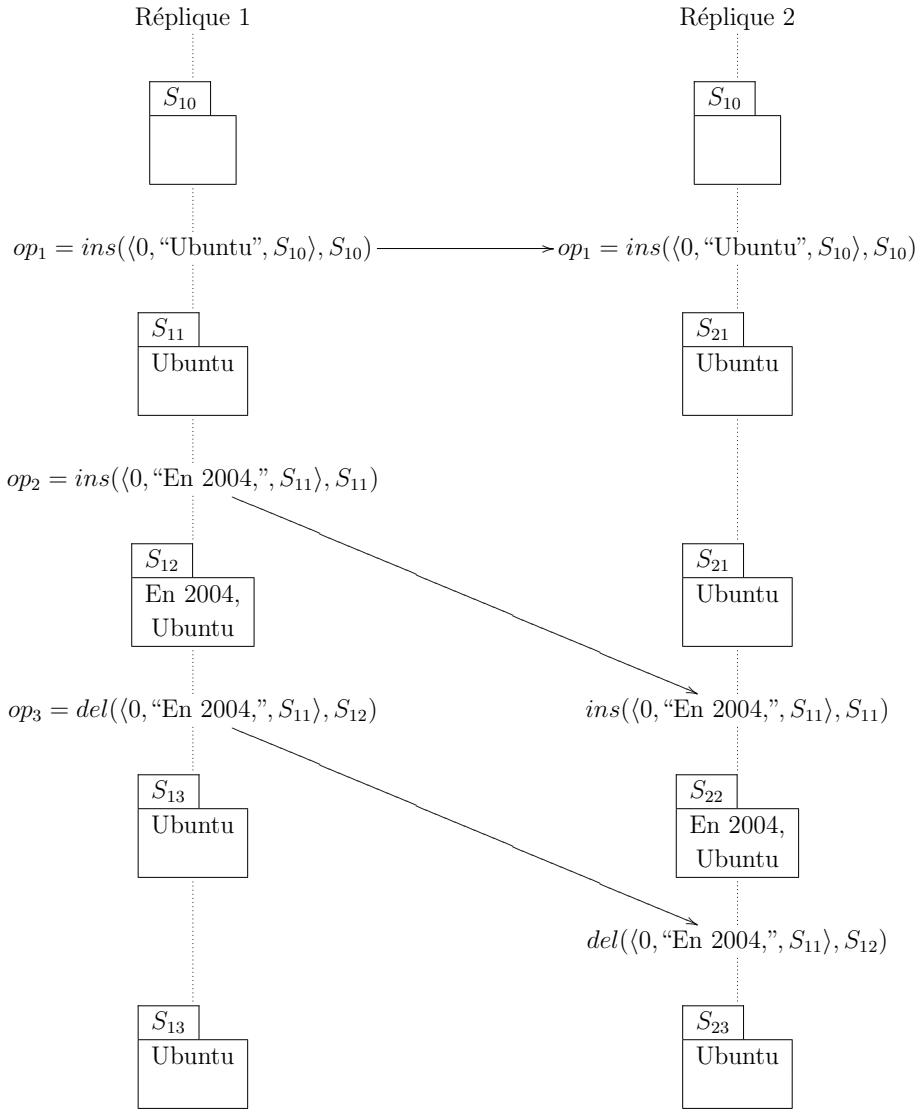


FIGURE 3.2 – Respect de la relation de précédence

Le respect de la précédence causale est un critère requis dans de nombreux systèmes distribués. Dans notre modèle, un mécanisme de réception causale peut être modélisé par des contraintes sur la livraison via le prédicat *Livrable*.

3.3.2 Convergence des répliques

Dans un système de réplication optimiste, les opérations ne sont pas forcément reçues dans le même ordre sur toutes les répliques, même si la causalité est respectée. En effet, deux opérations peuvent être *concurrentes* [Lam78], c.-à-d., aucune des deux ne précède causalement l'autre.

Définition 15 (Concurrence)**.** *Soit op_1 et op_2 deux opérations. Si $\neg(op_1 \rightarrow op_2)$ et $\neg(op_2 \rightarrow op_1)$ alors op_1 et op_2 sont dites concurrentes. On note alors $op_1 \mid\mid op_2$.*

La Figure 3.3 illustre une situation de concurrence. Les opérations op_4 et op_5 ont été générées sur les répliques 1 et 2. Intuitivement, deux opérations sont concurrentes si elles ont été générées en « même temps », c.-à-d., si chacune a été générée avant la réception de l'autre.

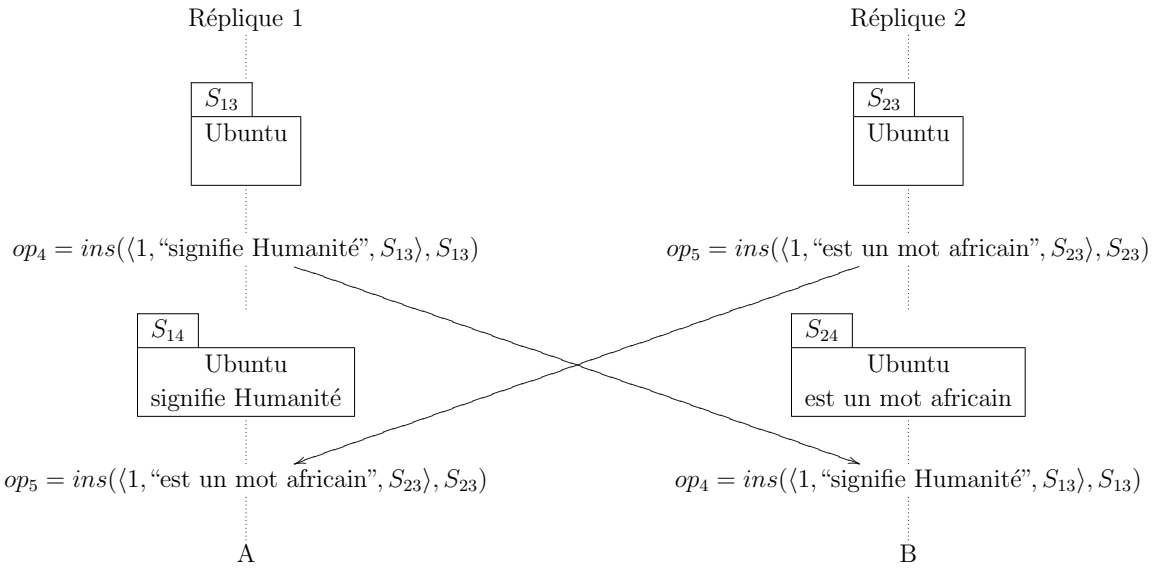
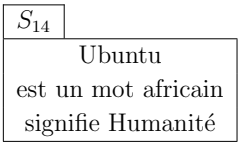


FIGURE 3.3 – Exécution concurrente

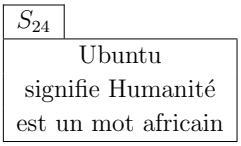
Supposons que 2 utilisateurs u_1 et u_2 modifient ce document en parallèle(Figure 3.3). u_1 crée une opération op_4 alors qu'en parallèle u_2 insère une ligne au même endroit avec une opération op_5. Ces deux opérations sont concurrentes.

Imaginons maintenant, un système dans lequel l'exécution de l'insertion d'un élément dans un document ne dépendrait que de sa position d'insertion initiale. Dans cet exemple, il existe donc deux ordres d'exécution possibles :

* Si le système exécute op_4 puis op_5 on obtiendrait alors un document :



* Si l'ordre d'exécution est op_5 puis op_4, on obtiendrait le document :



Suivant l'ordre de réception, on obtient deux valeurs différentes pour le document. Les deux répliques n'ayant plus le même contenu, ce système n'assure pas la convergence.

Sun *et al.* [SJZ+98] définissent le critère de convergence par :

> « Lorsque le même ensemble d'opérations a été exécuté sur toutes les répliques, toutes les répliques du document partagé ont la même valeur. »

La notion de valeur est spécifique à chaque système d'édition collaborative.
Dans notre modèle, le critère de convergence est défini par :

Définition 16 (Convergence)**.** *Le système d'édition collaboratif assure la convergence si et seulement si :* $\forall S = (H_S, \rightarrow_S, n) \in \mathbb{S}.\forall S' = (H_{S'}, \rightarrow_{S'}, n') \in \mathbb{S}.(H_S = H_{S'} \Rightarrow D(S) = D(S'))$.

Cette définition signifie que, quel que soit l'ordre de réception des opérations, toutes les répliques ayant reçu les mêmes opérations ont le même contenu. Le critère de convergence ne donne pas d'indication sur la valeur vers laquelle le système converge. Ainsi, dans notre exemple, le système converge si et seulement $A = B$. Par exemple, l'état obtenu peut être l'un des ordres possibles, l'état initial, voire même un document vide.

3.3.3 Préservation de l'intention

Sun *et al.* [SJZ+98] définissent l'intention d'une opération O par :

> « L'intention d'une opération O est l'effet obtenu par l'application de O sur l'état du document à partir duquel O a été générée. »

Contrairement aux critères « respect de la causalité » et « convergence », il n'existe pas de définition formelle générique de l'intention. Elle doit être définie pour chaque système d'édition collaborative. Nous proposons une définition de l'intention dans les systèmes d'édition collaborative de texte basés sur des opérations d'insertion et de suppression. [3] On modélise l'intention, notée I, par une relation d'ordre strict partiel sur les éléments du document.

Définition 17 (Intention)**.** *On note* $I(S) = (E, \prec)$*, avec* $E \subseteq \mathbb{E}$ *et* $\prec \subseteq E \times E$ *étant une relation d'ordre strict partiel sur* E*, la fonction représentant les ordres partiels induits par les opérations associées à l'état* S*.*

D'une manière générale, l'exécution d'une opération ne doit pas modifier l'ordre des éléments présents. En effet, dans notre modèle, il n'est pas possible de changer l'ordre des éléments. Il est bien sûr possible de supprimer un élément et d'insérer un élément avec le même contenu à une autre position. Cependant, dans notre modèle, il s'agit d'un nouvel élément.

Nous définissons l'intention d'une opération de suppression :

Définition 18 (Intention d'une opération de *suppression*)**.** *Soit une réplique dans un état* $S \in \mathbb{S}$*, avec pour intention* $I(S) = (E, \prec)$*. Après exécution d'une opération de suppression* $Supp(e, S')$*, l'intention devient* $I(S \circ Supp(e, S')) = (E', \prec')$ *avec* $E' = E \setminus \{e\}$ *et* $\prec' = \prec \setminus \{(a, b) \mid a = e \vee b = e\}$*.*

3. Il existe peu de systèmes proposant la mise à jour d'un élément [SXSC04], de plus, l'intention d'une telle opération n'a jamais été définie.

Cette définition précise simplement qu'un élément supprimé doit être retiré du document, ainsi que les relations d'ordre qui lui sont associés.

Nous définissons l'intention d'une opération d'insertion :

Définition 19 (Intention d'une opération d'*insertion*)**.** *Soit une réplique dans un état $S \in \mathbb{S}$ avec pour intention $I(S) = (E, \prec)$. Soit $I(S' \circ Ins(e, S')) = (E', \prec')$ l'intention après génération d'une opération d'insertion.*

Après exécution de cette opération $Ins(e, S')$ sur la réplique dans l'état S, l'intention devient $I(S \circ Ins(e, S')) = (E \cup \{e\}, \prec \cup(\prec' \cap(E \times \{e\} \cup \{e\} \times E)))$.

Cette définition précise que l'élément inséré par l'opération sur une réplique doit apparaître sur toutes les répliques et, l'ordre entre les éléments, commun à l'état de génération $S' \circ Ins(e, S')$ et à l'état d'exécution S, doit être conservé.

Le critère de préservation de l'intention est respecté si :

> « Pour toute opération O, les effets de l'exécution de O sur chaque réplique sont similaires à l'intention de O, et les effets de l'exécution de O ne modifient pas les effets des opérations indépendantes. »

Finalement, dans notre modèle, nous définissons la préservation de l'intention :

Définition 20 (Préservation de l'intention)**.** *Un système d'édition collaborative de texte préserve l'intention si on a :*

$$\forall S \in \mathbb{S}, (E, \prec) = I(S) \wedge (E_S, \prec_S) = D(S) \Rightarrow (E = E_S) \wedge (\prec \subseteq \prec_S)$$

Cette définition précise que le document du système collaboratif D doit être compatible avec l'intention I, c'est-à-dire, le document doit contenir les mêmes éléments et l'ordre des éléments du document doit être une linéarisation de l'ordre partiel induit par les intentions.

Par exemple, en considérant l'exemple de la Figure 3.3 : Sur la réplique 1, l'opération op_4 insère une ligne après la ligne insérée par l'opération op_2 (voir Figure 3.2). L'ordre entre ces deux lignes doit être préservé aussi longtemps que ces deux lignes apparaissent dans le document. On a alors, $I(S_{14}) = (E, \prec)$ avec $E = \{$"En 2004,", "signifie Humanité"$\}$ et $\prec = \{$("En 2004,", "signifie Humanité")$\}$. Lors de la réception de l'opération op_5, on a $I(S \circ op_5) = \{E', \prec'\}$ avec $E' = E \cup \{$"est un mot africain"$\}$ et

$$\prec' = \{(\text{"En 2004,"}, \text{"signifie Humanité"}), (\text{"En 2004,"}, \text{"est un mot africain"})\}$$

Dans cet exemple, $\prec'$ ne précise pas l'ordre entre les éléments "signifie Humanité" et "est un mot africain". Par conséquent, un système d'édition collaborative générant un document $D(S_{14} \circ op_5) = (E'', \prec'')$ avec $E'' = E'$ et

$$\prec'' = \prec' \cup \{(\text{"signifie Humanité"}, \text{"est un mot africain"})\}$$

respecte l'intention pour l'état $S_{14} \circ op_5$.

Annulation Sun [Sun00] définit l'effet de l'annulation par :

> « Soit un document dans l'état :
>
> $$S_n = S_0 \circ [O_1, \ldots, O_{i-1}, O_i, O_{i+1}, \ldots, O_n]$$
>
> [S_n est l'état du document obtenu en exécutant les opérations O_i $(1 \leq i \leq n)$ à partir de l'état initial S_0.] L'effet de l'application de $Annuler(O_i)$ sur l'état S_n est la transformation de l'état S_n en un nouvel état S_{n-1} qui peut être exprimé par
>
> $$S_{n-1} = S_0 \circ [O_1, \ldots, O_{i-1}, O'_{i+1}, \ldots, O'_n]$$
>
> avec O'_x, $x \in \{i+1, i+2, \ldots, n\}$, l'opération que le système aurait exécuté, à la place de O_x, si O_i n'avait pas été exécutée avant. »

Cette définition implique que le document doit retourner dans l'état qu'il aurait atteint, si l'opération annulée n'avait été produite. Suivant les applications, certaines opérations peuvent dépendre sémantiquement d'opérations précédentes. Nous avons proposé de modéliser ces relations via la relation $Sem(op_1, op_2)$. Dans la définition de l'annulation, ces dépendances sont annulées implicitement, en effet, si O_x dépend de de O_i, alors l'opération O'_x est une opération sans effet.

On note $Dep(op) \subset \mathbb{H}$ l'ensemble des opérations dépendant sémantiquement de l'opération op. On peut définir l'ensemble des dépendances sémantique d'une opération à partir de la relation $Sem(op_1, op_2)$. On a alors $Dep(op) = \{op_x \in \mathbb{H} \mid Sem^*(op, op_x)\}$, avec Sem^* la fermeture transitive de Sem. Dans notre modèle, l'intention revient à :

Définition 21 (Intention d'une opération d'annulation)**.** *Soit* $S = (H_S, \rightarrow_S, R_S)$

$$\forall op \in \mathbb{H}.A \subseteq H_S \Rightarrow I((H_S, \rightarrow_S, R_S)) = I((H_S \setminus A, \rightarrow_S \setminus \{(x,y) \mid x \in A \vee y \in A\}, R_S))$$

avec $A = \{op, Annuler(op, S)\} \cup Dep(op)$.

L'annulation d'une opération implique donc l'annulation des opérations qui en dépendent. Ces dépendances sont spécifiques au système considéré. Par exemple, dans le cas d'un éditeur de texte, nous pouvons considérer la dépendance suivante :

$$\forall e \in \mathbb{E}, Dep(Ins(e, S)) = \{Supp(e, S')\}$$

Un autre exemple consiste à considérer que l'insertion d'un élément entre deux autres éléments est dépendante de l'insertion de ces éléments :

$$\begin{aligned} \forall e \in \mathbb{E}.Dep(Ins(e,S)) \quad = \quad & \{Supp(e, S''')\} \cup \{Ins(e'', S'') \mid \exists S' \in \mathbb{S}.D(S') = (E, \prec).e'' \in E \wedge \\ & (((e, e'') \in \prec \wedge \nexists e'.e \prec e' \wedge e' \prec e'') \vee \\ & ((e'', e) \in \prec \wedge \nexists e'.e'' \prec e' \wedge e' \prec e))\} \end{aligned}$$

Il est bien sûr possible de « rejouer » une opération annulée. Dans ce cas, la réplique annule l'annulation, c.-à-d., génère une opération $Annuler(Annuler(op, S'), S)$ pour rejouer une opération op. La Figure 3.4 illustre l'intention de l'annulation. On considère deux répliques ayant reçu les mêmes opérations : $H_{S_{13}} = H_{S_{23}} = \{op_1, op_2, op_3\}$. Supposons que l'utilisateur sur la première réplique annule l'opération $op_2 = ins(\langle 0, \text{“En 2004,”}, S_{11}\rangle, S_{11})$.

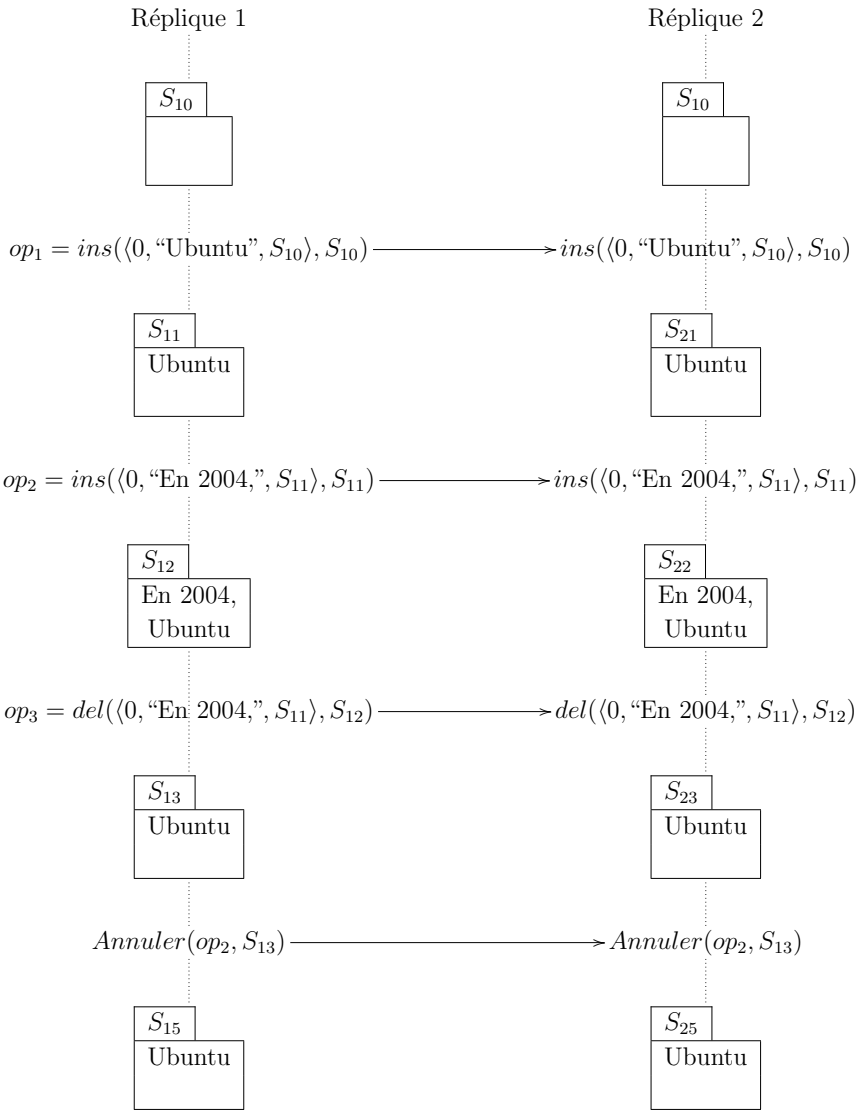


FIGURE 3.4 – Annulation

Par définition, le document sur la réplique 1 doit être dans l'état

$$I((\{op_1, op_2, op_3, Annuler(op_2, S_{13})\}, \rightarrow_{S_{15}}, R_1)) = I((\{op_1\}, \rightarrow_{S'_{15}}, R_1)) = I(S_{11})$$

.

avec

$$op_1 \rightarrow_{S_{15}} op_2 \rightarrow_{S_{15}} op_3$$

et

$$\rightarrow_{S'_{15}} = \emptyset$$

Dans cet exemple, on a $Dep(op_2) = \{op_3\}$ par conséquent $A = \{op_2, op_3, Annuler(op_2, S_{13})\}$.

Chapitre 4

Logoot

Sommaire

4.1 Logoot : un CRDT pour les documents texte

Dans ce chapitre, nous présentons « Logoot », un algorithme destiné à l'édition de documents texte sur réseau pair-à-pair. Notre proposition doit donc vérifier les critères du modèle de cohérence CCI tels que formalisés dans la section 3.3.1. Les approches d'édition collaborative sur réseau pair-à-pair existantes se basent sur l'utilisation de pierres tombales (voir section 2.2). Malheureusement, le nombre de pierres tombales ne peut que grossir au cours de l'édition et réduit donc le passage à l'échelle des approches existantes. Dans ce chapitre, nous présentons notre solution « Logoot » qui ne repose pas sur l'utilisation de pierres tombales.

Nous commençons par présenter notre approche, son modèle de données et son algorithme avant d'en discuter la complexité. Nous poursuivons par la démonstration de la vérification des critères du modèle de cohérence CCI, puis nous discutons du passage à l'échelle de notre solution. Finalement, nous terminons ce chapitre par la présentation des résultats de l'expérimentation que nous avons menée afin de valider notre approche.

4.1.1 Modèle de données de Logoot

Un document Logoot est composé de *lignes* définies par : $\langle idl, content\rangle$ où *content* est le contenu de la ligne et *idl* un *identifiant de ligne* non mutable et unique dans le temps et l'espace. On suppose l'existence de deux lignes $\langle idl_0, l_B\rangle$ et $\langle idl_\infty, l_E\rangle$ qui représentent le début et la fin du document. Les identifiants de ligne sont ordonnés suivant la relation d'ordre total $\prec$. Un document Logoot est donc un ensemble ordonné $(L, \prec)$ avec L l'ensemble des lignes possibles.

Par un exemple, un document Logoot peut ressembler à :

1 $\langle idl_0, l_B\rangle$
2 $\langle idl_1$, "Ceci est un exemple de document Logoot" $\rangle$
3 $\langle idl_2$, "Ici, $idl_1 \prec idl_2$ " $\rangle$
4 $\langle idl_3$, "Et $idl_2 \prec idl_3$ " $\rangle$
5 $\langle idl_\infty, l_E\rangle$

Dans cet exemple, on a par définition, $idl_0 \prec idl_1 \prec idl_2 \prec idl_3 \prec idl_\infty$.

Pour insérer une ligne dans un document Logoot, nous devons générer un identifiant de ligne correspondant à la position d'insertion souhaitée. Par exemple, si un utilisateur souhaite insérer une ligne entre les lignes $\langle idl_2$, "Ici, $idl_1 \prec idl_2$ " $\rangle$ et $\langle idl_3$, "Et $idl_2 \prec idl_3$ " $\rangle$, nous devons associer à la nouvelle ligne un identifiant idl_4 tel que $idl_2 \prec idl_4 \prec idl_3$.

De plus, puisqu'un utilisateur peut toujours insérer une ligne, nous devons être capable de produire une position A telle que $P \prec A \prec N$ pour tout P et N (tels que $P \prec N$). Par conséquent, l'ensemble des identifiants Logoot doit être un ensemble ordonné et dense.

La définition suivante présente un identifiant de ligne vérifiant ces caractéristiques. Nous supposons que chaque réplique gère une horloge logique c incrémentée à chaque création d'une nouvelle ligne.

Définition 22 (Identifiant de ligne et position)**.** *Une* position *est un triplet* $\langle i, s, c\rangle$ *où* $i \in [0, BASE[$ *est un chiffre dans la base* $BASE$*,* s *un identifiant unique de la réplique et* c *la valeur de l'horloge logique de cette réplique* s*.*

Un identifiant de ligne *est une liste ordonnée non-mutable de positions. On note* $p = p_0.p_1...p_n$ *un identifiant de ligne dont les positions sont les* p_i *avec* $0 \leq i \leq n$*. Soit* $\langle i_n, s_n, c_n\rangle$ *la dernière position d'un identifiant de ligne ,* s_n *est l'identifiant de la réplique qui a créé la ligne et* c_n *la valeur de son horloge à la création de cette ligne.*

La longueur d'un identifiant de ligne *correspond au nombre de positions qui la composent. On note* $\mid p \mid$ *la longueur d'un identifiant de ligne* p*.*

On définit aussi la distance entre deux identifiants de ligne $\overline{(p, q)}$ *comme étant le nombre d'identifiants* x *tels que* $p \prec x \prec q$ *et* $\mid x \mid = max(\mid p \mid, \mid q \mid)$*. Il s'agit donc du nombre d'identifiants compris entre* p *et* q *d'une longueur n'excédant pas le plus long identifiant parmi* p *et* q*.*

Les chiffres i des triplets « position » peuvent être choisis dans n'importe quelle base. Nous ne faisons pas de supposition sur cette base, elle doit, cependant, être la même sur toutes les répliques. Par exemple, dans notre validation (voir section 4.3), nous choisissons la base 2^{64}, la taille standard d'un entier long non signé. Selon la définition ci-dessus, les identifiants de ligne sont uniques dans le temps et l'espace. En effet, la dernière position de chaque identifiant contient l'identifiant unique et l'horloge de la réplique l'ayant généré. Deux répliques ne peuvent donc pas générer le même identifiant. De même, l'horloge étant incrémentée à chaque génération, une même réplique ne peut générer deux fois le même identifiant. Pour obtenir un ordre total entre les identifiants, nous utilisons la définition suivante.

Définition 23 (Ordre total strict pour les identifiants de ligne et les positions)**.**

- *Soit* $p = \langle i_1, s_1, c_1\rangle$ *et* $q = \langle i_2, s_2, c_2\rangle$ *deux positions, nous avons* $p <_{id} p$ *si et seulement si* $i_1 < i_2$*, ou* $i_1 = i_2$ *et* $s_1 < s_2$*, ou* $i_1 = i_2$*,* $s_1 = s_2$ *et* $c_1 < c_2$*,*
- *Soit* $P = p_0.p_1 \dots p_n$ *et* $Q = q_0.q_1 \dots q_m$ *deux identifiants de ligne, nous avons* $P \prec Q$ *si et seulement si* $\exists j \leq m.\ (\forall i < j.\ p_i = q_i) \wedge (j = n + 1 \vee p_j <_{id} q_j)$*.*

Enfin, le modèle de document Logoot est illustré figure 4.1. Les identifiants $\langle 0, NA, NA\rangle$ et $\langle MAX, NA, NA\rangle$ — où $MAX = BASE - 1$ — sont des identifiants de ligne spéciaux marquant le début et la fin du document. Ces identifiants sont respectivement, plus petit et plus grand que tous les identifiants de ligne d'après la définition 23.

Les identifiants peuvent être stockés dans une structure séparée, que nous appelons « table des identifiants ». L'absence de pierre tombale permet d'obtenir un coût de correspondance entre le document D et la table des identifiants constant. En effet, pour récupérer l'identifiant de la $i^{\text{ème}}$ ligne, il suffit de lire le $(i + 1)^{\text{ème}}$ identifiant de la table.

Nous avons maintenant défini notre modèle de données, les sections suivantes traitent de la modification de ces données.

Table des identifiants	Document
$\langle 0, NA, NA\rangle$	
$\langle 131, 1, 0\rangle$	"Voici un document Logoot"
$\langle 131, 1, 0\rangle.\langle 2471, 5, 23\rangle$	"Cette ligne se trouve entre 131 et 131"
$\langle 131, 3, 2\rangle$	"Cette ligne était la 2ème générée par la réplique 3"
$\langle MAX, NA, NA\rangle$	

FIGURE 4.1 – Modèle de données

4.1.2 Modification d'un document Logoot

Nous supposons que les utilisateurs peuvent modifier le document en y ajoutant ou en supprimant des lignes. Chaque opération effectuée sur le document implique une modification de la table des identifiants associée.

La suppression d'une ligne dans le document nécessite la suppression l'identifiant associé à cette ligne. De même, l'insertion d'une ligne dans le document nécessite d'insérer un identifiant pour cette ligne à la même position dans la table des identifiants. Cette insertion peut être faite avec un coût amorti constant [BCD+99] en utilisant une structure de données comme, par exemple, la classe « java.util.ArrayList » [Mic04].

On remarque que la suppression d'une ligne ne requiert pas l'insertion d'une pierre tombale contrairement à l'approche Woot. En effet, dans Woot, l'ordre entre les éléments est partiel. En supprimant un élément, on perd une partie de l'ordre. Les pierres tombales sont donc utilisées pour conserver cet ordre. Dans Logoot, on utilise un ordre total, il devient donc possible de supprimer un élément.

Quand une ligne est créée entre une ligne avec un identifiant x et une ligne avec un identifiant y, un nouvel identifiant z est calculé. Pour préserver l'intention selon la définition 3.3.3, nous devons respecter l'ordre suivant : $x \prec z \prec y$. Afin de construire z, nous pouvons choisir n'importe quelle stratégie : entre x et y au hasard, au centre de x et y, ou à proximité de x ou de y ... *De plus, une réplique peut choisir arbitrairement sa stratégie respectant les intentions. Une réplique peut également modifier sa stratégie à tout moment et sans avertir les autres répliques.* Dans cette section, nous présentons deux stratégies pour construire z.

Création des identifiants de ligne

Dans de nombreux systèmes d'édition collaborative tels que les wikis, Google Wave ou les logiciels de gestion de configuration, une modification sur un document n'est pas constituée d'une opération unique, mais d'un ensemble d'opérations appelé « patch ». Les lignes insérées par un patch sont souvent contiguës ou groupées dans des blocs contigus. Ainsi, pour répartir les identifiants de ligne d'un bloc de taille N, nous définissons la fonction generateLineIdentifiers (p, q, N, s) qui génère sur une réplique s, les N identifiants compris entre un identifiant de ligne p et un identifiant de ligne q. L'algorithme utilise des nombres écrits en base BASE. Il est possible qu'il n'y ait pas assez d'identifiants

de longueur 1 disponibles entre p et q. Dans ce cas, nous augmentons la longueur des identifiants : on dit que la longueur des identifiants de ligne croit. Dans le pire cas, chaque identifiant généré a $log_{BASE}(N)$ positions de plus que le plus long des identifiants parmi p et q.

Pour obtenir des identifiants de ligne courts, l'algorithme sélectionne d'abord le plus petit indice *index* tel que les préfixes de longueur *index* des positions p et q soient espacés d'au moins N.

Nous proposons deux stratégies pour générer les N identifiants :

- Stratégie aléatoire : les N identifiants sont générés aléatoirement parmi tous les identifiants de longueur *index* compris entre p et q,
- Stratégie boundary : tout comme la stratégie aléatoire, les identifiants sont générés aléatoirement mais deux identifiants successifs ne peuvent être espacés de plus d'une valeur notée *boundary*.

Stratégie aléatoire

Nous proposons la définition suivante pour la fonction generateLineIdentifiers dans le cas de la stratégie aléatoire.

```
1  function generateLineIdentifiers(p, q, N, rep)
2
3    let list := ∅,
4      index := 0,
5      interval := 0;
6    while (interval < N)
7      index++;
8      interval := prefix(q, index) − prefix(p, index) − 1;
9    endwhile
10   let step := interval/N,
11     r := prefix(p, index);
12   for j:=1 to N do
13     list.add(constructIdentifier(r + Random(1, step), p, q, rep));
14     r := r + step;
15   done
16   return list;
17 end;
```

La fonction prefix(p, i) retourne un *nombre* dans la même base que les chiffres des « positions ». Les chiffres du nombre prefix(p, i) sont les i premiers chiffres de p (complétés avec des 0 si $|p| < i$). Ligne 8, l'algorithme calcule le nombre d'identifiants disponibles entre les deux préfixes.

Par exemple, supposons que $BASE = 100$ et qu'une réplique identifiée par s ayant une valeur d'horloge c souhaite insérer N lignes entre deux lignes identifiées par $p =<$

$2, 4, 7 >< 59, 9, 5 >$ et $q =< 10, 5, 3 >< 20, 3, 6 >< 3, 3, 9 >$. L'algorithme generateLineIdentifiers cherche les plus petits préfixes de même longueur de p et q espacés d'au moins N.

1. $prefix(p, 1) = 2_{100}$ et $prefix(q, 1) = 10_{100}$, il y alors 7 (c.-à-d., $10 - 2 - 1$) identifiants disponibles entre les préfixes de longueur 1,
2. $prefix(p, 2) = 2.59_{100}$ et $prefix(q, 2) = 10.20_{100}$, il y a alors 760 (c.-à-d., $1020 - 259 - 1$) identifiants disponibles entre les préfixes de longueur 2,
3. $prefix(p, 3) = 2.59.0_{100}$ et $prefix(q, 3) = 10.20.3_{100}$, il y a alors 76102 (c.-à-d., $102003 - 25900 - 1$) identifiants disponibles entre les préfixes de longueur 3, etc.

Finalement, l'algorithme génère aléatoirement (ligne 13) les N identifiants parmi les *interval* identifiants disponibles. La randomisation permet de réduire la probabilité que deux répliques génèrent simultanément le même chiffre, et donc de réduire le taux de croissance des identifiants de ligne. La fonction constructIdentifier(r, p, q, rep) est définie ci-dessous.

```
1  function constructIdentifier(r, p, q, rep)
2
3    let id := {};
4    for i:=1 to |r| do
5      d := i^th position of r
6      if (d = p_i.digit) then
7        s := p_i.repid; c := p_i.clock
8      elseif (d = q_i.digit) then
9        s := q_i.repid; c := q_i.clock
10     else
11       s := rep.identifier; c := rep.clock++
12     fi;
13     id := id . ⟨d, s, c⟩;
14   done;
15   return id;
16 end;
```

Par exemple, supposons que la réplique identifiée par s avec une valeur d'horloge c souhaite insérer N lignes entre deux lignes identifiées par $p = \langle 2, 4, 7\rangle$ et $q = \langle 10, 5, 3\rangle\langle 20, 3, 6\rangle$ avec $BASE = 100$.

- Si $N < 8$, la fonction $generateLineIdentifiers(p, q, N, boundary, s)$ retourne les N identifiants répartis dans l'ensemble $\{\langle x, s, c\rangle \mid x \in]2, 10[\}$,
- Si $8 \leq N < 761$, les N identifiants retournés appartiennent à l'ensemble
  $\{\langle 2, 4, 7\rangle\langle x, s, c\rangle \mid x \in [0, BASE[\}\cup$
  $\{\langle y, s, c\rangle\langle x, s, c\rangle \mid y \in]2, 10[, x \in [0, BASE[\}\cup$
  $\{\langle 10, 5, 3\rangle\langle x, s, c\rangle \mid x \in [0, 20[\}$
- etc.

Stratégie « Boundary »

La stratégie « Boundary » répartit les identifiants construits de façon aléatoire en utilisant *boundary* pour limiter la distance entre deux identifiants successifs.

```
1  function generateLineIdentifiers(p, q, N, boundary, rep)
2
3    let list := {},
4      index := 0,
5      interval := 0;
6    while (interval < N)
7      index++;
8      interval := prefix(q, index) − prefix(p, index);
9    endwhile
10   let step := min(interval/N, boundary),
11     r := prefix(p, index);
12   for j:=1 to N do
13     list.add(constructIdentifier(r + Random(1, step), p, q, rep));
14     r := r + step;
15   done
16   return list;
17 end;
```

Le paramètre « boundary » permet de réduire la distance entre les identifiants de ligne et donc, de laisser un espace libre pour des insertions futures. Contrairement à la stratégie aléatoire, cette stratégie tient compte du fait que les insertions ont plus généralement lieu à la fin du document qu'au début. Une fois les préfixes sélectionnés, l'algorithme construit aléatoirement les N identifiants. On obtient pour l'exemple précédent, avec $boundary = 10$:

1. si $0 < N \leq 7$, les identifiants sont répartis dans l'ensemble $\{\langle x, s, c\rangle \mid x \in]2, 10[\}$,
2. si $7 < N \leq 760$ les identifiant sont alors générés parmi l'ensemble des 760 identifiants disponibles : $\{\langle 2, 4, 7\rangle\langle x, s, c\rangle \mid x \in]59, 100[\}$
   $\cup \{\langle y, s, c\rangle\langle x, s, c\rangle \mid y \in]2, 10[, x \in [0, 100[\}$
   $\cup \{\langle 10, 5, 3\rangle\langle x, s, c\rangle \mid x \in [0, 20[\}$.
3. etc.

La distance entre les identifiants de deux lignes successives ne dépasse pas *boundary*. Dans l'exemple ci-dessus, avec $N = 23$, les identifiants de ligne sont répartis dans l'ensemble des $N * boundary = 230$ premiers identifiants (c.-à-d., entre p et $\langle 4, s, h\rangle\langle 89, s, h\rangle$). En comparaison, la stratégie aléatoire génère les $N = 23$ identifiants parmi 760 identifiants (voir figure 4.2).

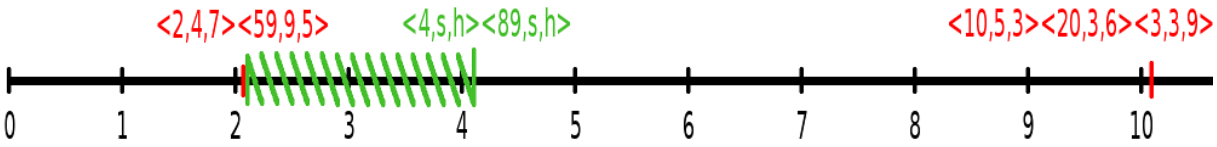


FIGURE 4.2 – Espaces de génération des identifiants. En hachuré, l'espace utilisé avec la stratégie « boundary ».

Propagation des changements et intégration des modifications distantes

Une opération est créée pour chaque modification sur le document. Pour chaque insertion, nous créons une opération *Insert(id, content)* où *id* est l'identifiant de la ligne et *content* le contenu de la ligne insérée. De même, nous créons une opération *Delete(id)* pour chaque ligne supprimée du document. Les opérations sont regroupées dans des patches qui sont envoyés à toutes les autres répliques afin d'y être intégrées. Ainsi, pour chaque opération reçue, une réplique doit à la fois mettre à jour la table des identifiants et le document. La fonction deliver définie ci-dessous est chargée d'intégrer un patch.

```
1  function deliver(patch)
2
3    for op in patch do
4      switch (op):
5      case Insert(id, content):
6            position := binarySearch(IdTable, id);
7            insertDoc(position, content);
8            insertIdTable(position, id);
9
10     case Delete(id):
11           position := binarySearch(IdTable, id);
12           if(IdTable[position] = id) then
13              deleteDoc(position);
14              deleteIdTable(position);
15           fi
16     end;
17   done
18 end;
```

L'exécution des opérations d'insertion et de suppression peut être faite dans un temps logarithmique en fonction du nombre de lignes dans le document et constant en fonction du nombre d'utilisateurs dans le réseau pair-à-pair. En effet, nous utilisons tout simplement l'algorithme de recherche dichotomique (lignes 6 et 11) pour trouver la position dans la table de l'identifiant de ligne le plus petit supérieur ou égal à l'identifiant *id*. Une fois la position trouvée, nous insérons ou supprimons le contenu à cette position dans le

document. De plus, l'intégration d'une opération de suppression peut supprimer la ligne du document, étant donné que l'ordre total entre les lignes restantes n'est pas affecté. L'algorithme vérifie uniquement que la ligne n'a pas déjà été supprimée (ligne 12) par une opération de suppression concurrente.

Notre approche suppose que les opérations seront reçues dans un ordre causal. En effet, les exécutions des opérations d'insertion et de suppression de la même ligne ne commutent pas, la convergence ne serait donc pas garantie. En effet, si une réplique exécute l'insertion puis la suppression, la ligne n'est plus dans le document. Par contre, si une réplique exécute d'abord la suppression, qui est sans effet puisque la ligne n'est pas dans le document, puis l'insertion, l'état final contient la ligne. La convergence n'est donc pas assurée. Pour assurer une réception causale, on peut utiliser n'importe quelle approche décrite dans la section 2.1.1.

4.1.3 Analyse en complexité moyenne

Supposons que la longueur d'un identifiant de ligne est en moyenne k. L'algorithme de génération cherche la plus petite valeur possible de l'*index*. Pour chaque incrément, on calcule un intervalle. Ainsi, en supposant que l'intervalle est calculé progressivement, la complexité temporelle moyenne de l'algorithme de génération est $O(k)$.

L'algorithme d'exécution utilise l'algorithme de recherche dichotomique pour trouver la position. Par conséquent, cet algorithme requiert en moyenne $log(n)$ comparaisons, où n est le nombre de lignes dans le document, et chaque comparaison coûte $O(k)$. Par conséquent, la complexité temporelle est $O(k.log(n))$.

La longueur des identifiants affecte aussi la complexité en espace. En effet, la table des identifiants contient un identifiant par ligne, par conséquent, la complexité en espace est $O(k.n)$.

La complexité des approches basées sur des pierres tombales (WOOT et MOT2) est impactée par la présence des pierres tombales dans le document. En effet, pour trouver un identifiant ou une position, ils doivent parcourir en moyenne la moitié du modèle. Ainsi, ils ont une complexité fonction de N, le nombre d'éléments dans le modèle. N est égal à n plus le nombre de pierres tombales.

	Génération	Intégration		Complexité
	Insertion/Suppression	Insertion	Suppression	en espace
Logoot	$O(k)$	$O(k.log(n))$	$O(k.log(n))$	$O(k.n)$
WOOT	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$
MOT2	$O(N)$	$O(N)$	$O(N)$	$O(N)$
TreeDoc	$O(K.log(N))$	$O(K.log(N))$	$O(K.log(N))$	$O(N)$

K est le nombre moyen de nœuds contenus dans un super-nœud (voir section 2.2.5).

k, la longueur des identifiants de ligne (bornée par N). Théoriquement, elle peut croître à chaque fois qu'une ligne est insérée. Par conséquent, dans le pire cas, nous avons $k = N$. Ainsi, si aucune ligne n'est jamais supprimée, la complexité en espace peut être $O(\sum^N i)$,

c'est-à-dire, $O(N^2)$.

4.2 Correction et passage à l'échelle

Dans cette section, nous montrons que Logoot respecte le modèle de cohérence CCI tout en supportant les caractéristiques des réseaux pair-à-pair.

4.2.1 Correction de l'approche

Puisque la correction des systèmes d'édition collaborative repose sur les critères CCI, nous devons vérifier que Logoot assure la convergence, préserve la causalité et l'intention.

La causalité

Pour assurer la causalité telle que définie section 3.3.1, nous pouvons utiliser n'importe quel mécanisme de diffusion causale.

La convergence

Pour assurer la convergence, le modèle CRDT suppose que les opérations concurrentes commutent. Si les identifiants de ligne sont uniques, non mutable, et totalement ordonnés, les différentes répliques peuvent exécuter une série d'opérations d'insertion dans n'importe quel ordre et d'obtenir le même résultat.

Lemme 1. *Les identifiants de ligne sont uniques (c.-à-d., on ne peut pas générer deux lignes avec le même identifiant sur la même réplique).*

Démonstration. La dernière *position* $\langle p, s, c\rangle$ de chaque identifiant de ligne contient l'identifiant unique s et la valeur de l'horloge c de la réplique qui a généré la ligne. Puisque les couples (s, c) sont uniques, les identifiants de ligne sont uniques. □

Théorème 2. *Si la causalité est respectée, Logoot assure la convergence.*

Démonstration. Nous devons vérifier que l'exécution de chaque couple d'opérations concurrentes commute.

- $(Insert(p, content_p), Insert(q, content_q))$: Les identifiants de ligne étant uniques (voir lemme 1), non mutables et totalement ordonnés, on a $p \prec q$ ou $q \prec p$. Quel que soit l'ordre d'exécution, on obtient le même document, les lignes « $content_p$ » et « $content_q$ » étant ordonnées suivant l'ordre entre p et q. Par conséquent, l'exécution de deux opérations $Insert(p, content_p)$ commute.
- $(Insert(p, content_p), Delete(q))$: la préservation de la causalité impose que p et q soient différents. La position d'insertion étant déterminée via un ordre total, la présence ou non de l'identifiant q ne modifie pas la position de l'identifiant p par

rapport aux identifiants différents de q présents dans le document. Par conséquent, les opérations $Insert(p, content_p)$ et $Delete(q)$ commutent.
- ($Delete(p)$, $Delete(q)$) : l'exécution d'une opération $Delete(p)$ retire uniquement l'identifiant p du document ainsi que la ligne correspondante. Si l'identifiant n'est pas dans le document, l'opération n'a pas d'effet. Par conséquent, deux opérations $Delete(p)$ commutent.

Par conséquent, chaque couple d'opérations commute. Logoot est un type de données CRDT et assure donc la convergence. □

Préservation de l'intention

Lemme 3. *Logoot respecte l'intention pour la suppression.*

Démonstration. Conformément à la définition de la préservation de l'intention (voir section 3.3.3) : « Une ligne supprimée ne doit pas apparaître dans le document tant que la suppression n'est pas annulée. ». L'exécution d'une opération *Delete* (voir section 4.1.2) supprime la ligne si elle ne l'a pas déjà été. □

Lemme 4. *Logoot respecte l'intention pour l'insertion.*

Démonstration. Étant donné que les identifiants sont non-mutables et totalement ordonnés, l'ordre des lignes observé sur la réplique générant l'insertion sera conservé sur les autres répliques. □

Théorème 5. *Logoot préserve l'intention.*

Démonstration. D'après les lemmes 3 et 4, Logoot assure l'intention de toutes les opérations considérées. En conséquence, Logoot préserve l'intention. □

4.2.2 Passage à l'échelle

Dans cette section, nous discutons du passage à l'échelle de notre approche suivant les 3 dimensions présentées dans la section 1.3.

La dynamicité

Dans Logoot, chaque réplique fonctionne de façon autonome. Logoot ne requiert pas la connaissance du nombre de répliques dans le réseau. Par conséquent, Logoot tolère les variations du nombre de répliques et passe donc à l'échelle en termes de dynamicité.

La symétrie

Dans Logoot, l'exécution d'opérations locales et distantes se fait sans concertation avec les autres répliques. En d'autres mots, chaque réplique décide seule du résultat, en

se basant uniquement sur les opérations qu'elle a reçues et sur son propre état. Il n'y a donc aucun élément central, ce qui permet à Logoot de passer à l'échelle en termes de symétrie.

Nombre d'utilisateurs et modifications

Comme l'illustre la section 4.1.3, les performances de Logoot ne dépendent pas du nombre de répliques. Elles dépendent uniquement du nombre de lignes dans le document et de la longueur des identifiants. Logoot passe à l'échelle en nombre de lignes si la longueur moyenne des identifiants k reste petite car sa complexité en temps et en espace dépend de k et du logarithme du nombre de lignes.

Remarque sur la causalité : Si Logoot passe à l'échelle sur les dimensions précisées précédemment, il requiert néanmoins un mécanisme de réception causale qui passe tout autant à l'échelle. Le passage à l'échelle de Logoot peut donc être limité par le mécanisme de réception causale. Afin de garantir une réception causale tout en passant à l'échelle nous pouvons utiliser un mécanisme de réception tel que celui décrit dans [KS98]. Nous pourrions également utiliser une diffusion probabiliste [EGH+03] en association avec une barrière causale [PRS97], dont la taille est inférieure à celle du vecteur d'horloge.

4.2.3 Conclusion

Dans cette section, nous avons présenté Logoot, un CRDT dédié à l'édition collaborative de documents texte. Cette approche a l'avantage de ne pas utiliser de pierres tombales. Nous avons proposé deux stratégies pour générer des identifiants les plus courts possibles. La première utilise tout l'espace disponible alors que la seconde groupe les identifiants au début de l'espace afin de laisser de la place pour des insertions futures. Malheureusement, la longueur d'un identifiant n'est bornée que par le nombre de modifications effectuées. Au pire cas, la complexité de Logoot varie linéairement avec le nombre de modifications. Cependant, nous émettons l'hypothèse que la longueur des identifiants est en pratique très faible. Dans la section suivante, nous présentons les résultats des expériences menées pour mesurer la longueur réelle d'un identifiant Logoot.

4.3 Expérimentations

Dans cette section, nous nous intéressons à la validation de notre approche. En effet, théoriquement, la longueur des identifiants de ligne Logoot peut atteindre N, le nombre d'insertions. Des identifiants d'une telle longueur remettent en question la validité de l'approche. Nous faisons l'hypothèse qu'en pratique, la longueur des identifiants est très inférieure à N. Afin de vérifier cette hypothèse, nous avons besoin d'un corpus d'édition collaborative. Un tel corpus n'étant, malheureusement, pas disponible, nous avons donc

fait le choix d'en constituer un à partir des modifications effectuées sur l'encyclopédie en ligne « Wikipédia » [Wik]. Dans la suite de cette section, nous détaillerons notre corpus, ainsi que les résultats obtenus avec notre algorithme, en fonction de la stratégie de génération des identifiants considérée.

4.3.1 Méthodologie

Dans un premier temps, nous décrivons très brièvement les détails de notre réalisation qui influent sur les résultats obtenus, puis, nous présentons le corpus que nous avons obtenu à partir de Wikipédia.

Réalisation

L'algorithme Logoot a été développé en Java dans le but de valider notre hypothèse. Nous utilisons une base $BASE = 2^{64}$ et donc un chiffre est représenté par un entier de 8 octets, par conséquent, une « position » dans Logoot est constituée de 20 octets : un entier pour le chiffre, l'identifiant de la réplique et 4 octets pour une horloge logique. Cette réalisation permet de choisir entre les stratégies « aléatoire » et « Boundary », afin de les comparer. Pour les mesures avec la stratégie « Boundary », nous avons arbitrairement fixé la valeur *Boundary* à 1 million. Logoot générant chaque identifiant à l'aide d'une fonction aléatoire, nous avons ré-exécuté les modifications dix fois afin d'obtenir des valeurs moyennes.

Présentation du corpus

Nous avons constitué notre corpus à partir de pages de la version anglaise de Wikipédia que nous groupons en deux catégories :

- Les pages extrêmes : cette catégorie contient les 10 pages spéciales les plus éditées, les 10 articles les plus édités et les 10 articles les plus longs. Ces 30 pages sont utilisées afin de vérifier le passage à l'échelle de notre approche dans les cas limites.
- Les pages « choisies aléatoirement » : Cette catégorie contient 2000 pages choisies aléatoirement. Ces pages sont utilisées pour illustrer le comportement de notre algorithme sur des cas moyens.

Les pages extrêmes

Les pages les plus éditées (table 4.1) ne sont pas des articles au sens « encyclopédique ». Elles peuvent être des pages de discussion, des forums ou des pages spéciales éditées principalement par des robots. Ces pages contribuent largement à la qualité des articles de Wikipédia, car elles servent à la communication et à la coordination entre les utilisateurs de Wikipédia. Le tableau suivant présente les 10 pages les plus éditées, avec leur nombre de « patches », c'est-à-dire, le nombre d'éditions pour chaque page. Dans la suite de cette

section, nous nous référerons à une page par le numéro que nous lui associons dans ce tableau. Par exemple, la « page 3 » fait référence à la page « Wikipedia:Sandbox ».

Id	pages	patches
1	Wikipedia:Administrator_intervention_against_vandalism	438330
2	Wikipedia:Administrators'_noticeboard/Incidents	343406
3	Wikipedia:Sandbox	170440
4	Wikipedia:Reference_desk/Science	142722
5	Wikipedia:Reference_desk/Miscellaneous	148283
6	User:Cyde/List_of_candidates_for_speedy_deletion/Subpage	152974
7	Wikipedia:WikiProject_Spam_LinkReports	149538
8	Wikipedia:Help_desk	126509
9	Wikipedia:Administrators'_noticeboard	138460
10	Template_talk:Did_you_know	91739

TABLE 4.1 – Les pages les plus éditées

Nous pouvons classer ces pages en quatre catégories :

- Forum : Il s'agit des pages 1, 2, 4, 5, 8 et 9. Dans ce type de pages, les utilisateurs ajoutent des lignes à la fin. Le début est régulièrement supprimé afin de limiter la taille des pages. Ce mode d'édition constitue le pire cas pour notre approche. En effet, les identifiants de ligne ne peuvent pas être réutilisé et des ajouts répétés en fin oblige à générer des identifiants toujours plus grand (au sens $\prec$) ce qui augmente la longueur des identifiants générés.
- Page éditée par un robot : Cyde, un administrateur de wikipédia, a créé un robot pour garder une trace des différentes pages proposées pour une suppression rapide. Ces pages sont ajoutées dans la catégorie « Category:Candidates for speedy deletion ». Malheureusement, dans Wikipédia, la liste des pages appartenant à une catégorie n'est pas « versionée », c'est-à-dire, il n'est pas possible de connaître la liste des pages ayant appartenu à une catégorie. Ce robot édite la page 6 afin de stocker ces versions. La page 7 est, quant à elle, éditée par COIBot, un robot, qui traque entre autres les liens de pourriel.
- Page de « suggestions » : Sur la page 10, les utilisateurs peuvent proposer des articles afin qu'ils apparaissent sur la page principale. Le mode d'édition de cette page est très particulier. Les propositions sont ajoutées au début de la page, alors que les anciennes propositions sont supprimées en fin de page. Il s'agit donc du comportement inverse de celui observé sur les pages « Forum ». Cependant, ce mode d'édition est similaire au pire cas pour notre approche. En effet, les identifiants de ligne ne peuvent être réutilisés et la génération d'identifiants toujours plus petit (au sens $\prec$) augmente leur longueur.

– Bac à sable : La page 3 est une page utilisée pour tester le mode d'édition de Wikipédia. Les utilisateurs sont donc autorisés à faire n'importe quelle modification. Le contenu de cette page est automatiquement remis à zéro toutes les 12 heures.

Les articles les plus édités (table 4.2) sont des articles au sens encyclopédique. Elles traitent principalement de sujets populaires et contiennent un très grand nombre de modifications. Par exemple, la page 11, l'article le plus modifié de Wikipédia contient plus 40000 modifications. Le tableau suivant présente les 10 articles les plus édités. Comme pour la catégorie précédente, nous désignons une page par le numéro que nous lui associons.

Id	pages	patches
11	George_W._Bush	41563
12	Wikipedia	28977
13	United_States	24781
14	Jesus	20271
15	Wii	19991
16	World_War_II	19547
17	Adolf_Hitler	19489
18	Britney_Spears	19244
19	Deaths_in_2008	19027
20	Michael_Jackson	18956

TABLE 4.2 – Les articles les plus édités

Les articles les plus longs (table 4.3) sont souvent des listes d'éléments. Le classement ci-dessous est assujetti à d'importants changements en raison de la politique de subdivision de Wikipédia. En effet, pour des raisons de lisibilité, les utilisateurs sont encouragés à découper les articles trop longs en plusieurs articles. Les pages suivantes sont peu éditées.

Les pages « choisies aléatoirement »

Ces pages ont été sélectionnées au hasard parmi les articles de la version anglaise de Wikipédia (voir table 4.4). Nous avons choisi aléatoirement 1000 articles parmi l'ensemble des articles. Par la suite, nous désignerons ces articles par l'expression « pages normales ». En moyenne, ces articles contiennent très peu d'éditions. En effet, Wikipédia étant un système collaboratif, nous observons aussi des versions « en cours » de rédaction. Nous avons sélectionné 1000 autres articles parmi les articles de qualité. [4] Nous désignerons par « articles de qualité » ce nouvel ensemble de pages. Les pages de qualité respectent

4. Il s'agit d'articles appartenant aux catégories « Good Articles » et « Featured Articles ».

Id	Page	patches
21	Line_of_succession_to_the_British_throne	3317
22	List_of_World_War_I_flying_aces	640
23	Timeline_of_United_States_inventions	3199
24	United_States_at_the_2008_Summer_Olympics	2314
25	List_of_college_athletic_programs_by_U.S._State	868
26	List_of_Brazilian_football_transfers_2008	752
27	Licensed_and_localized_editions_of_Monopoly	318
28	Austrian_legislative_election,_2008	1086
29	List_of_sportspeople_by_nickname	2332
30	List_of_mass_murderers_and_spree_killers_by_ number_of_victims	1172

TABLE 4.3 – Les articles les plus longs

les critères de qualité définis par les éditeurs de Wikipédia. À terme, toutes les pages de Wikipédia aspirent à appartenir à cette catégorie.

catégorie	Nb de pages	nb moyen de modifications
« pages normales »	1000	59,5
Good Articles	734	772,3
Featured Articles	266	1564,2
« Article de qualité » (GA+FA)	1000	982,9

TABLE 4.4 – Les pages « choisie aléatoirement »

On observe que le nombre de modifications est plus important dans les pages de qualité.

Extraction des données

Afin d'extraire ce corpus, nous utilisons l'API de MediaWiki [Med10] pour obtenir un fichier XML contenant plusieurs révisions d'une page de Wikipédia. Puis, en utilisant un algorithme de diff [Mye86], on calcule les modifications effectuées entre deux révisions. Ces actions sont effectuées par un logiciel appelé mediawikiRE [WEI10] que nous avons développé afin de mener ces expérimentations.

4.3.2 Validation

A partir du corpus décrit précédemment, nous rejouons les modifications dans un document Logoot et nous mesurons la longueur des identifiants. Dans cette section, nous montrons que notre approche est viable en vérifiant que la longueur des identifiants est

petite. Nous choisissons de mesurer la longueur des identifiants de ligne dans la version actuelle des documents. Afin de s'affranchir du vandalisme pouvant intervenir sur la dernière version du document, nous mesurons la longueur moyenne des identifiants d'une page sur les 100 dernières révisions. Nous présentons aussi la valeur maximale de la longueur d'un identifiant obtenue sur les 100 dernières révisions.

Les pages les plus éditées La table 4.5 montre la longueur des identifiants en nombre de positions pour les deux stratégies.

Page	Logoot : stratégie aléatoire		Logoot : stratégie « boundary »	
	Moyenne	Maximale	Moyenne	Maximale
1	1,0	1	1,0	1
2	4,1	7	1,1	2
3	1,0	1	1,0	1
4	26,1	29	1,0	1
5	62,7	66	1,0	2
6	1,0	1	1,0	2
7	8,8	23	1,0	1
8	5,3	9	1,0	1
9	5,3	7	1,2	2
10	6,2	9	19,5	27
Moyenne	12,2	15,3	2,9	4,0

TABLE 4.5 – Longueur des identifiants de ligne pour les pages les plus éditées

Pour les pages 2, 4, 5, 8 et 9, les éditions sont principalement des ajouts en fin du document. Des ajouts répétés à la même position représentent le cas défavorable pour notre approche. La stratégie Boundary a été proposée pour tolérer les ajouts en fin et donc offre de meilleurs résultats que la stratégie aléatoire.

Les articles les plus édités Pour les articles les plus édités, on constate que les deux stratégies donnent des résultats très similaires. Ces valeurs sont très proches de la longueur minimale d'un identifiant de ligne.

Les articles les plus longs Pour les articles les plus longs, la longueur maximale d'un identifiant ne dépasse pas 9 positions avec la stratégie aléatoire et 3 positions avec la stratégie « Boundary ». En moyenne, la longueur des identifiants de ligne ne dépasse pas 2 positions.

La table 4.8 résume les longueurs moyennes k pour les pages extrêmes et présente les valeurs pour les pages « prises au hasard ». Hormis pour les pages les plus éditées avec

Page	Logoot : stratégie aléatoire		Logoot : stratégie « boundary »	
	Moyenne	Maximale	Moyenne	Maximale
11	1,0	1	1,0	1
12	1,0	1	1,0	1
13	1,0	1	1,0	1
14	1,0	1	1,0	1
15	1,0	1	1,0	1
16	1,0	1	1,0	1
17	1,0	1	1,0	1
18	1,0	1	1,0	1
19	1,0	1	1,0	1
20	1,0	1	1,0	1
Moyenne	1,0	1	1,0	1

TABLE 4.6 – Longueur des identifiants de ligne pour les articles les plus édités

la stratégie aléatoire, k reste petit. On remarque que les pages « prises au hasard », ainsi que les articles les plus édités, obtiennent en moyenne la longueur minimale.

Nous avons donc validé notre approche en vérifiant que la longueur des identifiants reste petite pour un système d'édition collaborative. Nous nous intéressons uniquement à la longueur des identifiants, en effet, une longueur trop importante rendrait notre approche non viable. Par la suite, nous envisageons de mesurer d'autres paramètres tels que le coût en mémoire ou le temps d'exécution.

4.3.3 Comparaison avec les approches existantes

Afin de comparer notre approche, nous avons choisi d'utiliser les pages extrêmes du corpus présenté dans la section précédente. Nous nous concentrons uniquement sur la taille du modèle exprimée en pourcentage de la taille du document.

Présentation des approches

Nous considérons uniquement les approches respectant le modèle de cohérence CCI et satisfaisant au moins une contrainte pair-à-pair.

Logoot est utilisé dans les mêmes conditions que pour la section précédente, c'est-à-dire :

- nous avons ré-exécuté les modifications dix fois pour chaque page afin d'obtenir des valeurs moyennes.
- une position contient 20 octets,

Page	Logoot : stratégie aléatoire		Logoot : stratégie « Boundary »	
	Moyenne	Maximale	Moyenne	Maximale
21	1,0	1	1,1	2
22	1,0	1	1,7	3
23	1,0	1	1,0	2
24	1,0	1	1,0	1
25	1,0	1	1,0	1
26	3,7	9	1,9	2
27	1,0	1	1,2	2
28	1,4	4	1,6	3
29	1,0	1	1,0	2
30	1,0	1	1,0	1
Moyenne	1,3	2,1	1,3	1,9

TABLE 4.7 – Longueur des identifiants de ligne pour les articles les plus longs

Logoot	pages les plus éditées	Articles les plus édités	Articles les plus longs	Articles de qualité	pages normales
stratégie aléatoire	12.2	1.0	1.3	1.0	1.0
stratégie Boundary	2,9	1.0	1.3	1.0	1.0

TABLE 4.8 – Nombre moyen de positions par identifiant de ligne pour les pages considérées.

– nous avons arbitrairement fixé la valeur *Boundary* à 1 million.

Wooto associe un identifiant unique par ligne. Cet identifiant est composé d'un identifiant de réplique et d'une horloge logique. De plus, on associe un degré par ligne, ce qui implique un coût de 16 octets par identifiant. Dans cette approche, les identifiants créés ne sont jamais supprimés, par conséquent, le nombre d'identifiants est égal au nombre d'insertions effectuées sur la page. Par conséquent, le surcoût de Wooto est directement calculé à partir du nombre d'insertions dans la page.

TreeDoc utilise un arbre dans lequel chaque nœud contient un identifiant ainsi que le contenu de la ligne. Les auteurs précisent que le surcoût d'un nœud est de 26 octets [PMSL09]. Les nœuds feuilles pouvant être supprimés de l'arbre, nous avons implémenté TreeDoc afin de tenir compte de ce mécanisme. En revanche, nous ne tenons pas compte de la procédure « flatten » car elle requiert l'utilisation d'un consensus entre l'ensemble des répliques, ce qui n'est pas réaliste dans un environnement pair-à-pair. Il existe un mécanisme pour équilibrer l'arbre dans un environnement à large échelle [LPS09]. Cependant, les expériences suivantes ne tiennent pas compte de ce mécanisme.

MOT2 appartient à l'approche des transformées opérationnelles [Mic07]. Dans MOT2, la réconciliation des répliques se fait deux à deux. MOT2 requiert des fonctions de transformation vérifiant les deux propriétés TP_1 et TP_2. Nous avons donc évalué MOT2 en combinaison avec les fonctions TTF [OUMI06b]. Par conséquent, chaque ligne supprimée est remplacée par un caractère spécial. Ainsi, le surcoût est de 1 octet par ligne supprimée. Ainsi, nous calculons directement le surcoût de MOT2 à partir du nombre de lignes supprimées.

Par la suite, nous comparons la taille du modèle des approches ci-dessus avec la taille du document. Cette dernière est obtenue à partir du nombre de caractères qu'il contient.

Comparaison des surcoûts

La table 4.9 illustre les résultats obtenus pour les pages les plus éditées. On constate que les approches Logoot et TreeDoc ont un surcoût de l'ordre de la taille du document pour l'ensemble des pages considérées. Au contraire, les approches WOOT et MOT2, basées sur les pierres tombales, ont un surcoût pouvant atteindre respectivement 3175947% et 158796% (pages 3) du document. Ce tableau montre bien les limites des approches basées sur des pierres tombales.

Page	Logoot		WOOT	MOT2	TreeDoc
	aléatoire	« boundary »			
1	27,8	27,8	359412,5	17969,2	59,5
2	27,2	13,4	1311,3	65	13,2
3	20,2	20,2	3175947,3	158796,4	27,3
4	186,6	8,9	4313,7	215,2	26,6
5	520,4	10,3	9365,4	467,8	70,1
6	27,7	27,8	278281,1	13912,7	137,7
7	121,4	11,8	33955,4	1697,2	106,2
8	57,6	13	11583	578,5	22,2
9	37,9	9,6	10695,8	534,4	25,9
10	59,4	177,3	1906,1	94,7	28,2
Moyenne	108,6	32	388677,2	19433,1	51,7

TABLE 4.9 – Comparaison du surcoût en pourcentage pour les pages les plus éditées

La table 4.10 illustre les résultats obtenus pour les articles les plus édités. Les observations faites précédemment restent valables pour cette catégorie. Les approches Logoot et TreeDoc ont un surcoût très faible alors que les approches utilisant des pierres tombales ont un surcoût supérieur.

La table 4.11 illustre les résultats obtenus pour les articles plus longs. Contrairement aux catégories précédentes, toutes les approches considérées ont un surcoût faible. Ce

Page	Logoot		WOOT	MOT2	TreeDoc
	aléatoire	« boundary »			
11	8,3	8,3	24318	1215,5	11,2
12	15,3	15,3	30894,5	1544	20,3
13	8,3	8,3	7343,9	366,8	11,5
14	9,8	9,8	5223,9	260,7	14,6
15	8,7	8,7	4109,4	205	14,4
16	8,3	8,3	5960,6	297,6	11,3
17	8,5	8,5	3345,3	166,8	12,0
18	9	9	4197,1	209,4	13,8
19	16,6	16,6	1939,5	96,1	24,3
20	7,1	7,1	3486,2	174	11
Moyenne	10	10	9081,8	453,6	14,4

TABLE 4.10 – Comparaison du surcoût en mémoire pour les articles les plus édités

résultat s'explique par le faible nombre d'éditions effectuées sur ces pages. *On remarque que l'approche la plus efficace est MOT2.* On peut donc dire que, si le document contient peu d'éditions, il est préférable d'utiliser une approche utilisant des pierres tombales.

4.3.4 Conclusions

À partir de l'ensemble de l'expérience, nous déduisons que :

- La longueur des identifiants de Logoot reste faible en pratique.
- La stratégie « boundary » de Logoot a un surcoût légèrement supérieur à celui de la stratégie « aléatoire » pour les articles les plus longs et les plus édités. Par contre, son surcoût est très fortement inférieur pour les pages les plus édités. Finalement, en moyenne, la stratégie « boundary » génère un surcoût largement inférieur.
- Pour les trente pages considérées, le surcoût moyen est :
  - 132641% pour WOOT,
  - 6631% pour MOT2,
  - 50% pour Logoot avec la stratégie « aléatoire »,
  - 37% pour Treedoc,
  - 26% pour Logoot avec la stratégie « boundary ».

Cependant, les expérimentations que nous avons effectuées n'incluent pas la concurrence. En effet, nous ne disposons pas de corpus avec des modifications concurrentes. Cependant, il serait souhaitable de réaliser des expériences en tenant compte de la concurrence. Nous envisageons par la suite de constituer un corpus contenant des modifications concurrents.

Page	Logoot		WOOT	MOT2	TreeDoc
	aléatoire	« boundary »			
21	23,7	24,6	610,4	29,3	31,6
22	71,1	113,4	117,9	2,3	117,4
23	10	10	108	4,9	15,5
24	52,7	52,7	393,4	17	72,1
25	34,6	34,6	60,7	1,3	51,5
26	27	15	14,2	0,3	16,5
27	39,3	44,3	86,5	2,4	62,6
28	19,7	23,4	56,8	2,1	23,9
29	19,1	19,1	102,9	4,2	34,5
30	11,6	11,6	99,9	4,4	17,3
Moyenne	30,9	34,9	165,1	6,8	44,3

TABLE 4.11 – Comparaison du surcoût en mémoire pour les articles les plus longs

Chapitre 5

L'annulation dans les CRDTs

Sommaire

5.1 $CRDT^+$: Un CRDT d'annulation

L'annulation est une fonctionnalité essentielle des éditeurs collaboratifs. Malheureusement, définir un mécanisme d'annulation générique correct selon le modèle CCI est un problème difficile [Sun00, PK94, RG99, AD92]. Notamment, le mécanisme d'annulation a besoin de savoir combien de fois une opération a été annulée ou rejouée comme illustré par la figure 5.1.

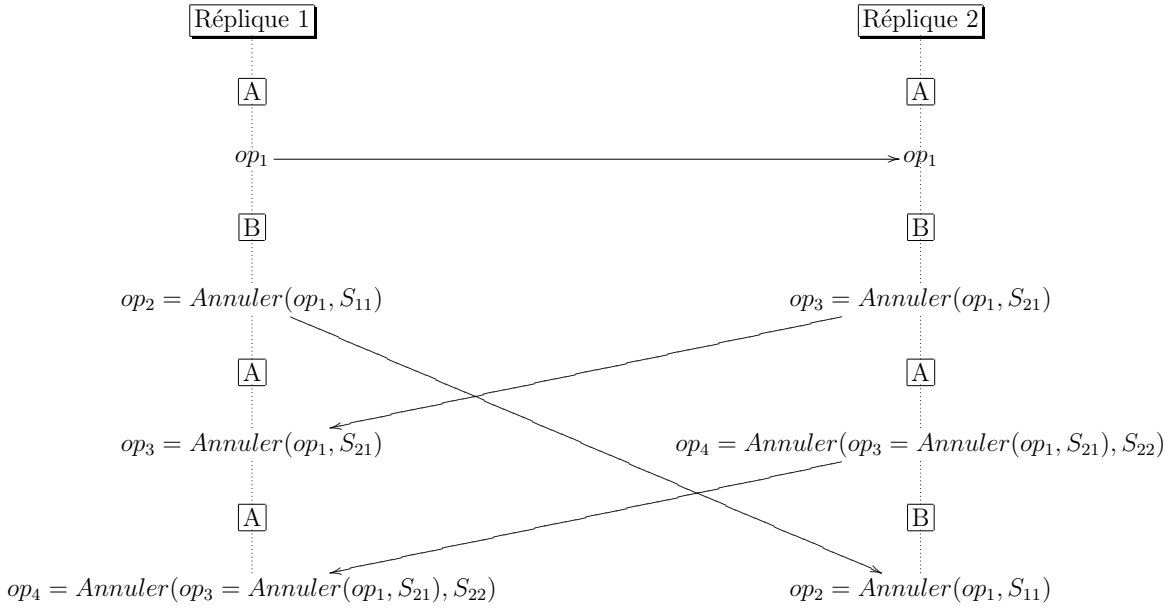


FIGURE 5.1 – Annulations concurrentes.

Supposons que deux répliques 1 et 2, aient pour état initial A. Après réception de la même opération op_1, elles atteignent l'état B. Ensuite, les deux répliques décident d'annuler cette opération en même temps. Par conséquent, la réplique 1 génère une opération $op_2 = Annuler(op_1, S_{11})$ alors que la réplique 2 génère $op_3 = Annuler(op_1, S_{21})$. Finalement, la réplique 2 choisit de rejouer l'opération op_1 : $op_4 = Annuler(Annuler(op_1, S_{21}), S_{22})$. Enfin, chaque réplique reçoit les opérations de l'autre. La réplique 1 a reçu deux « annulations », puis un message pour « rejouer » op_1. Si le mécanisme d'annulation sait uniquement que op_1 est annulée, la réception du message « rejouer » va ré-appliquer op_1. Malheureusement, cet exemple, viole la définition de l'intention d'une annulation (définition 21 du modèle section 3.3.3). En effet, si l'opération op_3 n'a jamais été produite, la seule opération active est op_2, op_1 doit rester annulée et le résultat final respectant l'intention doit être « A ».

Dans ce chapitre, nous proposons un mécanisme d'annulation générique pour les CRDTs fournissant une annulation correcte selon le modèle CCI (voir section 3.3). Nous construisons à partir d'un CRDT d'édition collaborative, un nouveau CRDT d'édition collaborative appelé $CRDT^+$ avec une fonctionnalité d'annulation. De plus, nous montrons que si le CRDT d'édition collaborative vérifie une propriété appelée « Inversibilité » alors,

le $CRDT^+$ assure le modèle CCI. Afin d'illustrer notre modèle, nous proposons alors une instance de $CRDT^+$ appelée « Logoot$^+$ » construite en utilisant le CRDT Logoot proposé dans le chapitre précédent. Nous proposons finalement une évaluation expérimentale de Logoot$^+$ afin de mesurer l'impact du mécanisme d'annulation sur les performances.

5.1.1 Hypothèses sur le CRDT

Dans notre modèle, nous supposons l'existence d'opérations inverses dans le CRDT d'édition collaborative, c'est-à-dire, pour une opération $op \in \mathbb{H}$, il existe une opération $\overline{op}$ telle que $\forall S \in \mathbb{S} \wedge Generable(op, S) \Rightarrow D(S \circ op \circ \overline{op}) = D(S)$. Ces opérations inverses sont utilisées afin d'obtenir l'annulation de l'effet d'une opération dans le document.

Définition 24 (Propriété d'inversibilité)**.**
Un système d'édition collaborative $\langle \mathbb{D}, \mathbb{H}, D, Livrable, Generable \rangle$ *respecte la propriété d'inversibilité si :*

1. *pour chaque opération, il existe une opération inverse :*

$$\forall op \in \mathbb{H}, St(op) \in \mathbb{S} \Rightarrow \exists \overline{op}. D(S \circ op \circ \overline{op}) = D(S)$$

   On note alors $\overline{\mathbb{H}}$ *l'ensemble des opérations inverses* $\overline{op}$*.*

2. *chaque opération inverse* $\overline{op}$ *commute avec toutes les opérations, à l'exception éventuelle de l'opération originale op et de ses dépendances* $Dep(op)$*.*

Par exemple, pour un éditeur de texte, l'inverse d'une opération *Insert(id, text, S)* peut être l'opération *Delete(id, text, S')*.

Dans notre modèle, nous proposons d'annuler toutes les opérations dépendantes de *op* avant d'annuler l'opération *op*. Par conséquent, l'opération $\overline{op}$ est toujours exécutée sur un état ne reflétant pas les opérations dépendantes de l'opération *op*. Finalement, il n'est pas nécessaire que l'opération $\overline{op}$ commute avec les opérations dépendantes de *op*.

Remarque : L'ensemble des opérations inverses $\overline{\mathbb{H}}$ n'est pas nécessairement le même que l'ensemble des opérations $\mathbb{H}$. Par exemple, dans la version de Woot avec annulation [RWSM$^+$09], nous avons défini trois opérations : *Insert(id,T,S)*, *Delete(id,T,S)* et *Undelete(id,T,S)*. L'opération *Insert(id,T,S)* n'est pas l'inverse de l'opération *Delete(id,T,S)* car elle n'insère que des identifiants *id* uniques et ne peut donc pas insérer un identifiant ayant déjà été généré. L'opération *Undelete* a donc été introduite comme l'inverse d'une suppression. Dans cet exemple, on a donc $\mathbb{H} = \{Insert(id, T, S), Delete(id, T, S)\}$ et $\overline{\mathbb{H}} = \{$*Delete(id,T,S)*, *Undelete(id,T,S)*$\}$.

5.1.2 Modèle du $CRDT^+$

A partir d'un CRDT $\langle \mathbb{D}, \mathbb{H}, D, Livrable, Generable \rangle$, notre approche permet d'obtenir un $CRDT^+$ $\langle \mathbb{D}^+, \mathbb{H}^+, D^+, Livrable^+, Generable^+, Annuler \rangle$. Dans cette section, nous

définissons les paramètres de notre $CRDT^+$: $\mathbb{H}^+$, $\mathbb{D}^+$, D^+, $Generable^+$ et $Livrable^+$ en fonction des paramètres du CRDT : $\mathbb{H}$, $\mathbb{D}$, D, $Generable$ et $Livrable$.

Opérations : Le $CRDT^+$ introduit une opération d'annulation notée $Annuler(op, S)$ avec $op \in \mathbb{H}^+$ l'opération à annuler et $S \in \mathbb{S}^+$ l'état de génération tel que $S = St(Annuler(op, S))$ de cette annulation.

L'ensemble des opérations $\mathbb{H}^+$ d'un $CRDT^+$ est défini par :

$$\mathbb{H}^+ = \{Annuler(op, S) \mid op \in \mathbb{H}^+ \wedge S \in \mathbb{S}^+\} \cup \mathbb{H}$$

avec $\mathbb{H}$ l'ensemble des opérations du système d'édition collaborative.

Documents : Le modèle $CRDT^+$ est composé d'un document appelé « document d'annulation » et du document du CRDT. Le document d'annulation est une collection d'opérations. [5] Pour chaque opération dans le document d'annulation, nous associons un entier que nous appelons degré. On a donc :

$$\mathbb{D}^+ = \{(A, D) \mid A \in \{(op, d) \mid op \in \mathbb{H}^+ \wedge d \in \mathbb{Z}\} \wedge D \in \mathbb{D}\}$$

Nous supposons qu'il n'existe pas de couples (op, d) et (op, d') dans A tel que $d \neq d'$. Le degré est un entier relatif utilisé afin de représenter l'état d'une opération, c'est-à-dire, si elle a un effet ou pas. Le degré d'une opération est supérieur ou égal à 1 pour une opération ayant un effet et est strictement inférieur à 1 pour une opération annulée.

Livraison : Nous supposons qu'une opération d'annulation est livrable si l'opération qu'elle annule a déjà été exécutée :

$$\forall S = (H_S, \rightarrow_S, R). \forall op \in H_S. Livrable^+(Annuler(op, S))$$

Pour les opérations du CRDT, les conditions de livraison restent les mêmes :

$$\forall S, \forall op \in \mathbb{H}. Livrable^+(op, S) = Livrable(op, S)$$

Génération : Dans notre modèle, on considère qu'une opération d'annulation $Annuler(op, S)$ est générable uniquement si op a été préalablement exécutée : [6]

$$\forall S = (H_S, \rightarrow_S, R). \forall op \in H_S. Generable^+(Annuler(op, S))$$

Pour les opérations du CRDT, on ne change pas les conditions de génération :

$$\forall S, \forall op \in \mathbb{H}. Generable^+(op, S) = Generable(op, S)$$

5. Nous ne faisons aucune hypothèse sur la structure de cette collection. Le document d'annulation peut être, par exemple, une table de hachage, une liste ordonnée, ou un graphe. Nous supposons simplement qu'il permet la récupération d'une opération à partir de son identifiant dans un délai raisonnable.

6. On peut ajouter comme contrainte que l'opération n'est annulable que si son degré est 1. Cette contrainte n'influence pas la correction du modèle.

Exécution : La fonction $D^+(S \circ op)$ est définie grâce à l'algorithme deliver ci-dessous :

```
1  function deliver(op)
2    op.degree = 1
3    if (op = Annuler(op', S))
4      undo(op')
5    else
6      for op_i in A
7          if ( Sem(op_i, op) and op_i.degree < 1)
8              op.degree--
9      if ( op.degree > 0 )
10         execute(op) // Exécution de op sur le CRDT original
11   A.addOperation(op) // Ajoute l'opération avec son degré dans A
```

deliver fait simplement la distinction entre les opérations d'édition et les opérations d'annulation. Si *op* est une opération d'édition, l'algorithme calcule le degré à partir des opérations d'annulation dont dépend *op*. Si le degré résultant est 1, la fonction execute(*op*) est appelée, elle a pour effet de livrer *op* au système d'édition collaborative. Si *op* est une opération d'annulation, on appelle la fonction undo(*op*).

Afin de calculer le résultat d'une annulation, nous définissons deux fonctions : undo et redo. La fonction undo permet d'annuler une opération, ce qui implique l'annulation de toutes les dépendances éventuelles avant l'exécution réelle de l'annulation. De même, la fonction redo « rejoue » l'opération avant de rejouer les éventuelles dépendances de cette opération.

Le degré d'une opération *op* est décrémenté à chaque exécution d'une opération $Annuler(op, S)$. Une opération *op* dont le degré passe de 1 à 0 est dite déjouée et son effet est donc retiré du document, ce qui consiste à :

- Annuler l'annulation de l'opération op' si $op = Annuler(op', S)$, c.-à-d., appeler la fonction redo(op'),
- Sinon, *op* est une opération d'édition et il faut :
  1. Annuler les opérations dépendantes de *op*. Pour cela, on appelle récursivement la fonction undo(op_i) pour chaque opération op_i telle que $Sem(op, op_i)$, c.-à-d., les opérations dépendantes directement de *op* (voir définition dans la section 3.3.3),
  2. Appliquer l'inverse de l'opération *op*.

```
1  function undo(op)
2  op.degree--
3  if( op.degree = 0 )
4    if( op = Annuler(op', S) )
5      redo(op')
6    else
7      for op_i s.t. Sem(op, op_i)
```

```
8         undo(op_i)
9     execute(op̄)
```

A l'inverse, une opération op dont le degré passe de 0 à 1 doit être rejouée et son effet doit être ajouté au document, ce qui consiste à :

- Annuler l'opération op' si $op = Annuler(op', S)$, c'est-à-dire, appeler la fonction undo(op'),
- Sinon op est une opération d'édition et il faut :
  1. Exécuter l'inverse de l'inverse de l'opération op,
  2. Annuler l'annulation des opérations dépendant de l'opération op, c'est-à-dire, appeler récursivement la fonction redo pour chaque opération dépendant de op.

```
1 function redo(op)
2 op.degree++
3 if( op.degree = 1 )
4   if( op = Annuler(op', S) )
5     undo(op')
6   else
7     execute(op̿)
8     for op_i.Sem(op, op_i)
9       redo(op_i)
```

Les fonctions undo et redo sont des fonctions récursives afin de parcourir l'ensemble des opérations dépendantes de l'opération à annuler. Par définition, l'ensemble des dépendances est un graphe acyclique, par conséquent, la fonction undo termine.

La figure 5.2 illustre la propagation des degrés des opérations du document d'annulation sur le scénario de la figure 5.1.

(a) Initialement, le document d'annulation de la réplique 1 contient uniquement l'opération op_1 avec un degré de 1.

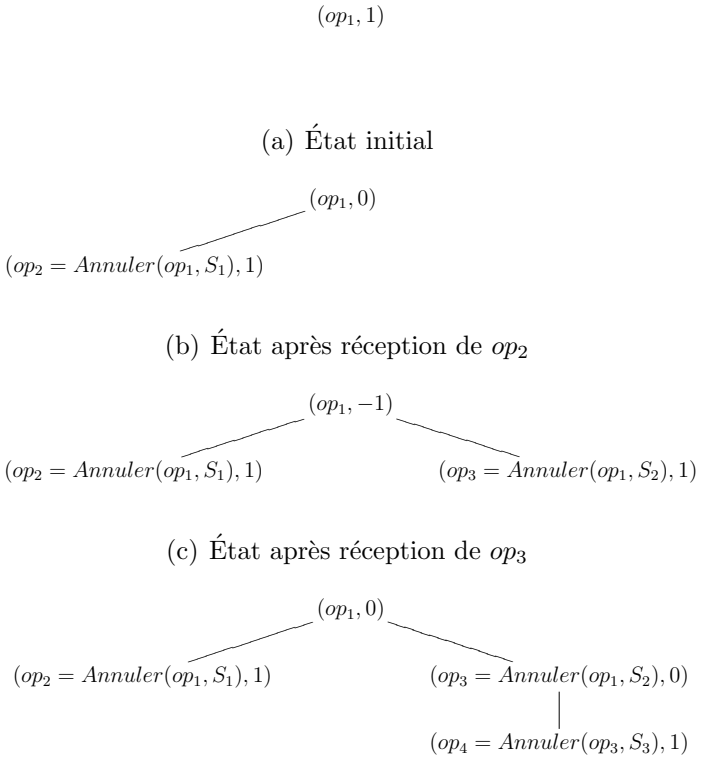
(b) A la réception de op_2 annulant op_1, nous exécutons la fonction undo(op_1). Le degré de l'opération op_1 est désormais de 0, et l'opération n'étant pas une opération d'annulation, on déjoue l'opération op_1.

(c) Similairement, la réception de op_3 décrémente le degré de op_1 qui devient -1.

(d) L'opération op_4 annule op_3, on décrémente donc le degré de l'opération op_3. L'opération op_3 n'a donc plus d'effet, il est donc nécessaire de retirer son effet : incrémenter le degré de l'opération op_1. Le degré de op_1 est maintenant 0, op_1 reste donc annulée. La réplique 1, figure 5.1, reste donc dans l'état A.

Correction

Un CRDT est une structure de données dans laquelle l'exécution des opérations concurrentes est commutative. Nous allons montrer qu'un $CRDT^+$ est un CRDT, c'est-à-dire,



(a) État initial

(b) État après réception de op_2

(c) État après réception de op_3

(d) État après réception de op_4

FIGURE 5.2 – Propagation des degrés

que l'exécution des opérations concurrentes commute sur le document d'annulation ainsi que sur le document du CRDT d'édition collaborative.

Théorème 6. *Un $CRDT^+$ construit à partir à partir d'un CRDT d'édition collaborative vérifiant la propriété d'inversibilité est un CRDT.*

Démonstration. Nous devons montrer que l'exécution des trois couples d'opérations concurrentes commute :

1. (op, op') : Par définition $op' \in Dep(op)$ implique $op \rightarrow op'$, donc op et op' ne sont pas concurrentes.

   Document d'annulation Si $op' \notin Dep(op)$, l'exécution de op' n'affecte pas le degré de l'opération op et réciproquement.

   Document d'édition En fonction du degré de chaque opération, elles seront exécutées ou non par le CRDT d'édition. Si elles sont toutes les deux exécutées, leur exécution commute car le document d'édition est un CRDT.

2. $(op', Annuler(op, S))$: Par définition, op et $Annuler(op, S)$ ne peuvent pas être concurrentes.

   Dans le cas où $op \neq op'$ et $op' \notin Dep(op)$:

Document d'annulation L'exécution de $Annuler(op, S)$ n'affecte pas le degré de l'opération op' et réciproquement.

Document d'édition Grâce à la propriété d'inversibilité, l'exécution de $Annuler(op, S)$ et op' commute.

Dans le cas où $op \neq op'$ et $op' \in Dep(op)$:

Document d'annulation nous allons montrer que les degrés des opérations op et op' sont les mêmes quel que soit l'ordre d'exécution. Si le degré de l'opération op est différent de 1, $Annuler(op, S)$ n'a aucun effet sur le degré des autres opérations, dont op'. L'exécution du couple ($Annuler(op, S)$,op') commute. Nous considérons désormais le cas où le degré de l'opération op est 1. Dans un premier temps, exécutons op' puis $Annuler(op, S)$:

(a) L'exécution de l'opération op' ajoute simplement le couple (op', $1 - n$) dans le document d'annulation (figure 5.3(b)), n étant égal au nombre d'opérations op_x telles que $Sem(op_x, op')$ et $op_x.degree < 1$

(b) L'exécution de l'opération $Annuler(op, S)$ commence par annuler les dépendances de op, soit op' dont le degré est désormais de $1 - n - 1$. Puis, l'algorithme annule op qui obtient un degré de $1 - 1 = 0$ (figure 5.3(c)).

Nous exécutons maintenant $Annuler(op, S)$ puis *op'* :

(a) L'exécution de l'opération $Annuler(op, S)$ annule toutes les opérations dépendant de op avant d'annuler op. Le degré de l'opération op est maintenant de $1 - 1 = 0$ (figure 5.4(b)).

(b) L'exécution de *op'* commence par calculer le degré (ligne 9 de la fonction deliver) de *op'* à partir de celui de op. On a $op'.degree = 1 - n - 1$ car le degré de op est inférieur à 1 (figure 5.4(c)).

Document d'édition L'exécution d'une opération d'annulation implique l'annulation de toutes les dépendances avant l'exécution de l'opération $\overline{op}$. Si on exécute op' d'abord, on obtient : $S \circ op \circ \ldots \circ op' \circ \ldots \circ \overline{op'} \circ \ldots \circ \overline{op} = S \circ op_0 \circ \ldots \circ op_n$ c'est-à-dire, seulement les opérations non dépendantes. Si on exécute $Annuler(op, S)$ en premier, l'exécution de op' n'aura pas d'effet sur le document d'édition, et on obtient $S \circ op \circ \ldots \circ \overline{op} = S \circ op_0 \circ \ldots \circ op_n$.

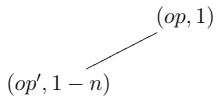
3. ($Annuler(op, S)$, $Annuler(op', S')$) : Par définition, $Annuler(Annuler(op, S), S')$ et $Annuler(op, S)$ ne peuvent pas être concurrentes.

Document d'annulation Les couples d'opérations ($Annuler(op, S)$, $Annuler(op', S')$) commutent trivialement car l'addition est une opération commutative.

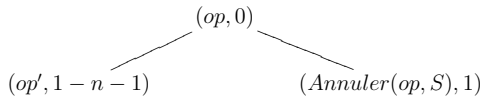
Document d'édition Dans le cas où $op' \notin Dep(op)$, la propriété d'inversibilité assure la commutativité. Sinon, dans les deux cas on obtient $S \circ op \circ \ldots \circ op' \circ \ldots \circ \overline{op'} \circ \ldots \circ \overline{op}$ car l'algorithme annule les dépendances (op') avant d'annuler l'opération op.

□

(a) État initial



(b) État après réception de op'



(c) État après réception de $Annuler(op, S)$

FIGURE 5.3 – Exécution de op' puis de $Annuler(op, S)$

Modèle de cohérence CCI

Dans cette section, nous montrons qu'un $CRDT^+$ est correct dans le modèle CCI (voir section 3.3) s'il est construit à partir d'un CRDT d'édition collaborative lui-même correct dans le modèle CCI et vérifiant la propriété d'inversibilité.

Préservation de la causalité : $CRDT^+$ n'impose pas de contrainte sur le respect de la causalité. Ainsi, nous devons respecter les mêmes contraintes que le système d'édition collaborative.

La convergence : La section 5.1.2 montre qu'un $CRDT^+$ est un CRDT. Un $CRDT^+$ assure donc la convergence si la causalité est préservée.

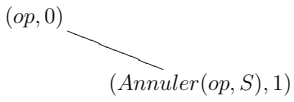
La préservation de l'intention : Les opérations d'édition op préservent l'intention du système d'édition par définition. Nous devons donc vérifier que les opérations $Annuler(op, S)$ respectent la définition de l'annulation (voir section 3.3.3).

Théorème 7. *Un $CRDT^+$ construit à partir d'un CRDT d'édition collaborative vérifiant le modèle CCI ainsi que la propriété d'inversibilité, préserve l'intention de l'annulation.*

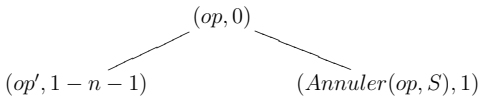
Démonstration. Sans perte de généralité, on suppose qu'une réplique vient d'annuler une opération et qu'il atteint un état S_x.

$$S_x = S \circ op \circ op_0 \circ \ldots \circ op_n \circ Annuler(op, S)$$

(a) État initial



(b) État après réception de op'



(c) État après réception de $Annuler(op, S)$

FIGURE 5.4 – Exécution de $Annuler(op, S)$ puis de op'

Parmi la séquence d'opérations suivant op, certaines dépendent de op, d'autres non. On note $Dep(op) = \{op_{d_j} \mid 0 \leq d_j \leq n \wedge 0 \leq j \leq m\}$ les $m + 1$ opérations dépendant de op.

On a alors,

$$D(S_x) = D(S \circ op \circ op_0 \circ \ldots \circ op_{d_m} \circ op_{d_m+1} \circ \ldots \circ op_n \circ \overline{op_{d_m}} \circ \ldots \circ \overline{op_{d_0}} \circ \overline{op})$$

La séquence d'opérations $op_{d_m+1} \circ \ldots \circ op_n$ ne contient aucune opération dépendant de op par définition. La propriété d'inversibilité garantit donc la commutativité entre cette séquence et l'opération $\overline{op_{d_m}}$.

$$D(S_x) = D(S \circ op \circ op_0 \circ \ldots \circ op_{d_m} \circ \overline{op_{d_m}} \circ op_{d_m+1} \circ \ldots \circ op_n \circ \overline{op_{d_m-1}} \circ \ldots \circ \overline{op_{d_0}} \circ \overline{op})$$

La propriété d'inversibilité nous donne :

$$D(S_x) = D(S \circ op \circ op_0 \circ \ldots \circ op_{d_m-1} \circ op_{d_m+1} \circ \ldots \circ op_n \circ \overline{op_{d_m-1}} \circ \ldots \circ \overline{op_{d_0}} \circ \overline{op})$$

En appliquant le même raisonnement pour chaque opération dépendante on obtient :

$$D(S_x) = D(S \circ op \circ \overline{op} \circ op_{a_0} \circ \ldots \circ op_{a_l})$$

avec $op_{a_0} \circ \ldots \circ op_{a_l}$ les $l + 1 = n - m$ les opérations de la séquence $op_0 \circ \ldots \circ op_n$ ne dépendant pas de l'opération op.

Finalement, en appliquant une dernière fois la propriété d'inversibilité, on obtient :

$$D(S_x) = D(S \circ op_{a_0} \circ \ldots \circ op_{a_l})$$

Par conséquent, si le CRDT vérifie la propriété d'inversibilité, le $CRDT^+$ préserve l'intention. □

5.1.3 La complexité temporelle du $CRDT^+$

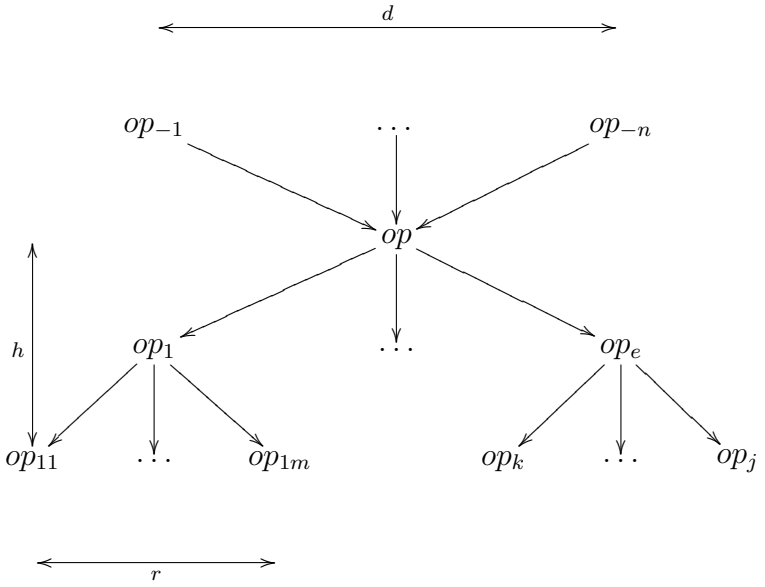


FIGURE 5.5 – Dépendances pour l'opération *op*.

On note d le nombre d'opérations maximum dont une opération peut dépendre. Par exemple, dans le cas d'un arbre, on peut avoir $d = 1$, c'est-à-dire, une opération d'insertion d'un nœud dépend uniquement de l'opération d'insertion du nœud père.

Une opération admet un ensemble d'opérations dépendantes. On note r le nombre de ces opérations. Par exemple, dans le cas d'un arbre binaire, si on considère qu'un nœud fils est dépendant de l'insertion du nœud père, l'opération d'insertion d'un nœud père admet au plus $2 * n$ opérations dépendantes d'insertion d'un nœud fils avec n le nombre de répliques dans le système d'édition. Il s'agit du cas particulier où chaque réplique ajoute en concurrence 2 fils à ce nœud.

d et r sont bornés par le nombre d'opérations produites N. On note h la plus longue chaîne de dépendance de l'opération *op*. Ces variables sont illustrées sur la figure 5.5.

La livraison d'une opération d'annulation fait appel aux fonctions undo et redo. Ces fonctions s'exécutent alternativement, le nombre d'appels correspond à la longueur de la « chaîne d'annulation », c'est-à-dire, le nombre d'emboîtements d'opérations d'annulation. Par exemple, $Annuler(Annuler(Annuler(op, S), S'), S'')$ corresponds à 3 emboîtements. On note u le nombre maximum d'emboîtements. u est borné par le nombre d'opérations N. Par conséquent, la complexité de la livraison d'une opération d'annulation est $O(u+r*h)$, soit dans le pire cas $O(N^2)$.

La livraison d'une opération d'édition implique le calcul du degré à l'aide des degrés des opérations dont elle dépend. Ainsi, la complexité de la livraison d'une opération d'édition est $O(d)$, soit dans le pire cas $O(N)$.

5.1.4 Optimisations pour les CRDTs sans dépendance

Dans cette section, nous montrons comment un $CRDT^+$ peut être optimisé si le système d'édition collaborative CRDT ne comprend pas de dépendance, c'est-à-dire :

$$\forall op \in \mathbb{H}.Dep(op) = \emptyset$$

Le document d'annulation

Une opération d'annulation se réfère à une opération précise, il est donc nécessaire d'associer un identifiant unique à chaque opération pouvant être annulée. Puisqu'on suppose que toutes les opérations peuvent être annulées, chaque opération, y compris les opérations d'annulation, doit avoir un identifiant unique. Malheureusement, une telle identification est coûteuse.

Cependant, un utilisateur souhaite rarement annuler une seule opération. Nous proposons de travailler avec des « patches », c'est-à-dire, un ensemble d'opérations successivement générées par une réplique.

Définition 25 (Patch)**.** *Dans un système d'édition collaborative, un patch généré par une réplique dans un état $S = \langle H_S, \rightarrow_S, n\rangle$ est défini par : $P = \langle H_p, \rightarrow_p\rangle$ avec $H_p \subseteq H$ tel que*

$$\begin{aligned}\forall op_i \in H_p \quad . \quad & St(op_i) = S \vee \\ & (\exists op_{i-1} \in H_p.St(op_i) = St(op_{i-1}) \circ op_i \wedge Generable(op_i, St(op_{i-1})))\end{aligned}$$

et

$$\rightarrow_p = (H_p \times H_p) \cap \rightarrow_S$$

On note $St(P)$ l'état de génération du patch et $\mathbb{P}$ l'ensemble des patches constitués des opérations du système d'édition collaborative.

Un patch est donc un ensemble atomique d'opérations pouvant être générées consécutivement par une réplique. Il est alors nécessaire de définir l'exécution d'un patch $P = \langle H_p, \rightarrow_p\rangle : D(S \circ P) = D(S \circ op_0 \circ \ldots \circ op_n)$ avec $op_i \in H_p$ et $op_0 \rightarrow_p \ldots \rightarrow_p op_n$.

Ainsi, l'annulation se fait sur les patches. Par conséquent, seuls les patches et les opérations d'annulation de patch nécessitent un identifiant unique.

Remarque : Cette solution ne peut pas être utilisée avec un système possédant des dépendances. En effet, annuler un patch revient à annuler les opérations dépendantes. Malheureusement, ces opérations dépendantes peuvent se trouver dans un patch avec d'autres opérations. En annulant les dépendances, on annulerait donc d'autre opérations.

Les algorithmes

L'absence de dépendance améliore grandement la complexité des fonctions undo, redo et deliver. En effet, les lignes 7 et 8 des fonctions undo et redo peuvent être retirées des algorithmes. De même, pour les lignes 6 à 10 de l'algorithme deliver qui sont spécifiques au traitement des dépendances.

Par conséquence, la complexité de la livraison d'une opération d'annulation devient $O(u)$ soit dans le pire cas $O(N)$. De même, la complexité de la livraison d'une opération d'édition passe de $O(N)$ à $O(1)$.

5.2 Logoot$^+$

Afin d'illustrer notre mécanisme d'annulation, nous avons construit un $CRDT^+$, appelé « Logoot$^+$ », à partir du CRDT d'édition collaborative Logoot décrit dans le chapitre 4. Nous considérons qu'il n'existe pas de dépendance entre les opérations de Logoot$^+$. Par conséquent, nous utilisons la notion de patch (Définition 25).

5.2.1 Logoot$^+$ et inversibilité

Dans cette section, nous montrons que Logoot ne satisfait pas immédiatement la propriété d'inversibilité et ne peut donc pas être directement utilisé pour obtenir un $CRDT^+$. Nous proposons alors des modifications et terminons par montrer que la nouvelle version de Logoot satisfait bien le critère d'inversibilité.

Logoot ne respecte pas l'inversibilité

La version de Logoot définie dans le chapitre précédent ne vérifie pas la propriété d'inversibilité. En effet, l'opération *Delete(pid, texte)* ne possède pas d'inverse. Par définition, l'opération *Insert(pid, texte)* possède un identifiant unique *pid*. Le modèle interdit la génération de plusieurs opérations avec le même identifiant. Cette restriction n'est pas uniquement un choix de conception mais elle est requise pour atteindre la convergence.

Afin d'illustrer cette divergence, supposons que deux répliques partagent un document contenant les lignes « A », « B » et « C » (voir figure 5.6). Les répliques 1 et 2 génèrent en concurrence respectivement les patches P1 et P2 pour supprimer ligne « B ». Puis, supposons que la première réplique ré-insère la ligne « B » avec le même identifiant. La réplique 1 reçoit alors P2, ce qui a pour effet de supprimer la ligne « B ». Finalement, la réplique 2 reçoit la suppression puis la ré-insertion, la ligne « B » apparaît donc donc le document.

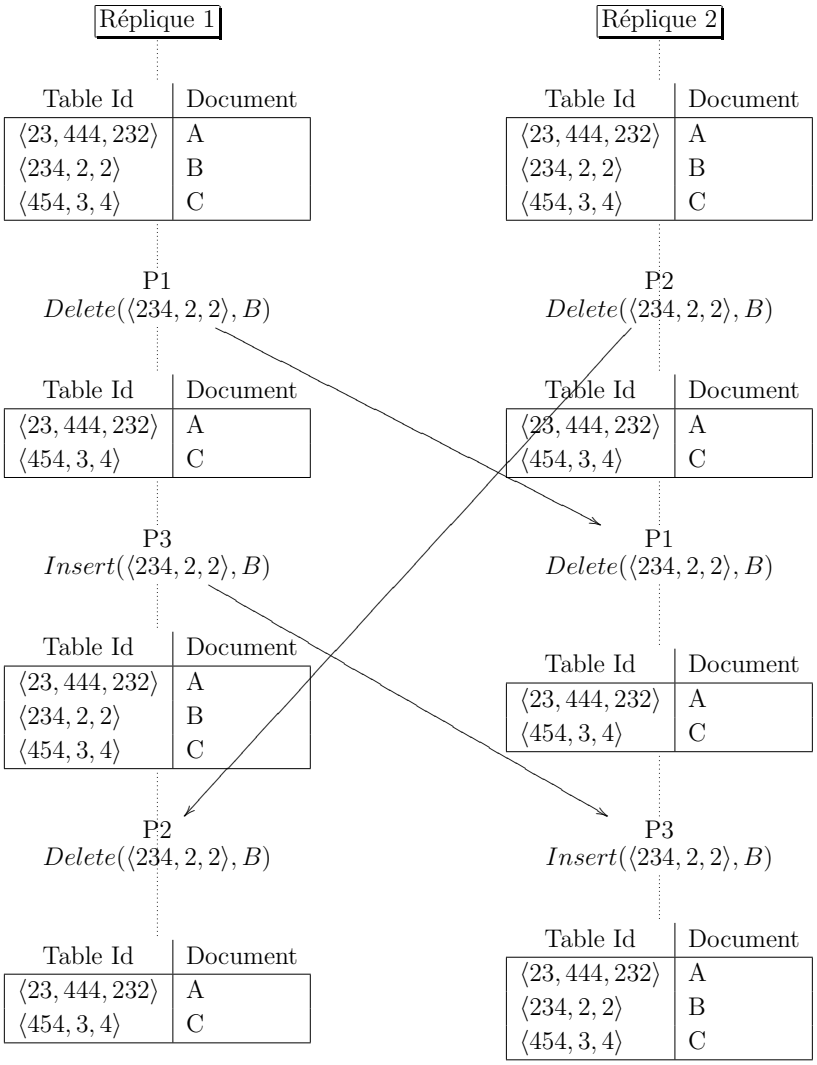


FIGURE 5.6 – Divergence avec les ré-insertions.

Extension de Logoot

Nous devons donc modifier Logoot afin qu'il vérifie la propriété d'inversibilité. Afin de garantir la convergence avec les opérations inverses, nous proposons une solution appelée « degré de visibilité ». Ce dernier a une fonction similaire au degré des opérations du $CRDT^+$. L'idée principale est de calculer pour chaque ligne un entier relatif représentant la visibilité de la ligne. Lors de la création de la ligne, nous affectons un degré de visibilité de 1. Chaque fois qu'une ligne est supprimée, nous décrémentons le degré de visibilité. De même, à chaque insertion, on incrémente le degré de visibilité. Une ligne avec un degré de visibilité supérieur ou égal à 1 est dite « visible » et apparaît dans le document. Sinon, la ligne est dite « invisible » et ne figure pas dans le document. Par exemple, sur la réplique 2 de la figure 5.6, la ligne « B » a un degré de visibilité de :

– 1 initialement
– 0 après le patch $P2$
– -1 après le patch $P1$
– 0 après l'exécution de l'opération $P3$

On suppose l'existence d'un mécanisme permettant de connaître le degré des lignes.

Théorème 8. *Logoot avec le degré de visibilité assure la propriété d'inversibilité.*

Démonstration. L'inverse de l'opération d'insertion *Insert*$(p, content_p)$ est l'opération *Delete*$(p, content_p)$. Similairement, l'inverse de l'opération de suppression *Delete*$(p, content_p)$ est l'opération *Insert*$(p, content_p)$. Les opérations *Insert*$(p, content_p)$ et *Delete*$(p, content_p)$ ayant pour unique effet d'incrémenter ou de décrémenter le degré de visibilité du même identifiant, l'exécution de ces opérations commute. Logoot avec le degré de visibilité assure la propriété d'inversibilité. □

5.2.2 Logoot$^+$, un algorithme de type $CRDT^+$

Logoot avec le degré de visibilité assurant la propriété d'inversibilité, nous pouvons l'utiliser pour construire un $CRDT^+$ et donc obtenir un mécanisme d'annulation.

L'ensemble des modifications est alors stocké dans notre document d'annulation. Puisque nous considérons l'absence de dépendance, nous pouvons grouper les opérations dans des patches. L'ensemble des opérations de Logoot$^+$ est $\mathbb{H}^+ = \{$ Insert$(p, content_p)$, Delete$(p, content_p)$, $Annuler(op, S)\}$ avec $op \in \mathbb{H}^+$.

Remarque : On suppose que les utilisateurs ne sont pas autorisés à générer directement deux insertions avec le même identifiant. En effet, un utilisateur ne peut générer un identifiant de ligne existant uniquement dans le cas de l'annulation d'une suppression. Par conséquent, le degré de visibilité d'un identifiant ne peut pas être supérieur à 1. En effet, si n répliques annulent la même suppression, le degré de visibilité ne sera incrémenté que de 1, d'après la définition de la fonction undo(op) (voir figure 5.1.2).

Calcul du degré de visibilité

Différentes solutions peuvent être utilisées pour obtenir le degré de visibilité d'une ligne. Une solution naïve consiste à parcourir le document d'annulation et à compter les opérations dans les patches non annulés. Un tel algorithme ne nécessite pas de stockage supplémentaire, mais a une complexité proportionnelle au nombre d'opérations produites : $O(N)$. Une autre solution est l'utilisation de pierres tombales [RWSM$^+$09]. Dans ce cas, le stockage a une complexité proportionnelle au nombre d'opérations produites : $O(N)$.

Afin de faciliter le calcul du degré de visibilité d'une ligne et d'obtenir une bonne complexité, nous proposons donc l'utilisation d'une structure de données appelée « Cimetière » contenant les degré de visibilité.

Le cimetière est une table de hachage dont les clés sont les identifiants de ligne et les valeurs les degrés de visibilité. Heureusement, la plupart des identifiant de ligne insérées ne sont pas présents dans le cimetière :

- chaque identifiant présent dans la table des identifiants a forcément un degré de visibilité de 1, par conséquent, nous ne stockons pas les identifiants présents dans la table des identifiants,
- les identifiants de ligne avec un degré de visibilité de 0 ne sont ni dans le cimetière ni dans la table des identifiants.

Par conséquent, le cimetière ne contient que les couples $(id, dvisibilite)$ où le degré de visibilité $dvisibilite$ de la ligne identifiée par id est strictement inférieur à 0, c.-à-d., les couples pour les lignes qui ont été supprimées en concurrence. En conséquence, nous prévoyons que le surcoût du cimetière en espace soit très faible dans les systèmes avec peu d'éditions concurrentes. Le cimetière a deux méthodes principales :

- cemeteryGet(id) : retourne 0 si l'identifiant id n'est pas dans le cimetière, sinon il retourne la visibilité associée à cette identifiant id,
- cemeterySet($id, dvisibilite$) insère le couple $(id, dvisibilite)$ dans le cimetière. Si l'identifiant id est déjà présent, l'ancienne valeur est écrasée. Si $dvisibilite = 0$, le couple est retiré du cimetière.

La fonction deliver(), présentée ci-dessous, est chargé de la gestion du degré de visibilité de la ligne. Elle applique l'effet d'une opération d'insertion lorsque le degré de la ligne passe de 0 à 1 et, similairement, elle applique les effets d'une opération de suppression lorsque le degré passe de 1 à 0.

```
1  function deliver(patch):
2    for op in patch do
3      switch (op)
4      case Insert(id, content, S):
5           dvisibilite := cemetery.get(id) + 1;
6           if (dvisibilite = 1) then
7             position := idTable.binarySearch(id);
8             document.insert(position, content);
9             idTable.insert(position, id);
10          else
11            cemetery.set(id, dvisibilite);
12          fi
13     case Delete(id, content, S):
14          position := idTable.binarySearch(id);
15          if (IdTable[position] = id) then
16            document.remove(position, content);
17            idTable.remove(position, content);
18            dvisibilite := 0;
19          else
```

	Génération		Intégration			Modèle
	Insert/Delete	Annulation	Insertion	Suppression	Annulation	
TreeDoc	$O(K.log(N))$	-	$O(K.log(N))$	$O(K.log(N))$	-	$O(N)$
Logoot	$O(k)$	-	$O(k.log(n))$	$O(k.log(n))$	-	$O(k.n)$
Logoot$^+$	$O(k)$	$O(N + k.log(n))$	$O(k.log(n))$	$O(k.log(n))$	$O(N + k.log(n))$	$O(N + k.n)$
WOOT	$O(N)$	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$	$O(N)$
MOT2	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$

TABLE 5.1 – Complexité temporelle moyenne de Logoot$^+$

```
20              dvisibilite := cemetery.get(id) − 1;
21            fi
22            cemetery.set(id, dvisibilite)
23      end;
24    done
25  end;
```

Enfin, la figure 5.7 illustre le comportement du cimetière. On suppose que les répliques 1 et 2 partagent le même document contenant trois lignes « A », « B » et « C ». Les deux répliques suppriment alors la ligne « B » en créant respectivement un patch P1 et un patch P2. Puis la réplique 1 annule le patch P1, l'opération inverse de la suppression est alors générée. Un patch P3 est alors appliqué. Lorsque la réplique 2 reçoit P1 contenant l'opération de suppression de la réplique 1, le degré de la ligne « B » devient -1 et l'identifiant de cette ligne est ajouté dans le cimetière. La réception de l'inverse de l'opération de suppression, augmente le degré de la ligne « B ». Le degré étant 0, l'identifiant est retiré du cimetière. Similairement, lorsque la réplique 1 reçoit le patch P2, le degré de la ligne « B » doit être décrémenté. Cette ligne étant dans le document, son degré passe de 1 à 0. La ligne est alors simplement retirée du document.

5.2.3 Analyse en complexité moyenne de Logoot$^+$

La complexité de Logoot$^+$ reste la même que celle de Logoot pour les opérations d'édition. Pour les opérations d'annulation, le $CRDT^+$ a une complexité moyenne de $O(u)$ et l'exécution d'une opération inverse à un coût de $O(k.log(n))$. Le coût d'une annulation est donc de $O(u + k.log(n))$.

k, la longueur des identifiants de ligne, est non bornée.

Passage à l'échelle

Logoot$^+$ passe à l'échelle en nombre d'utilisateurs et de modifications, ainsi qu'en termes de symétrie et de dynamicité pour les mêmes raisons que Logoot (Voir section 4.2.2).

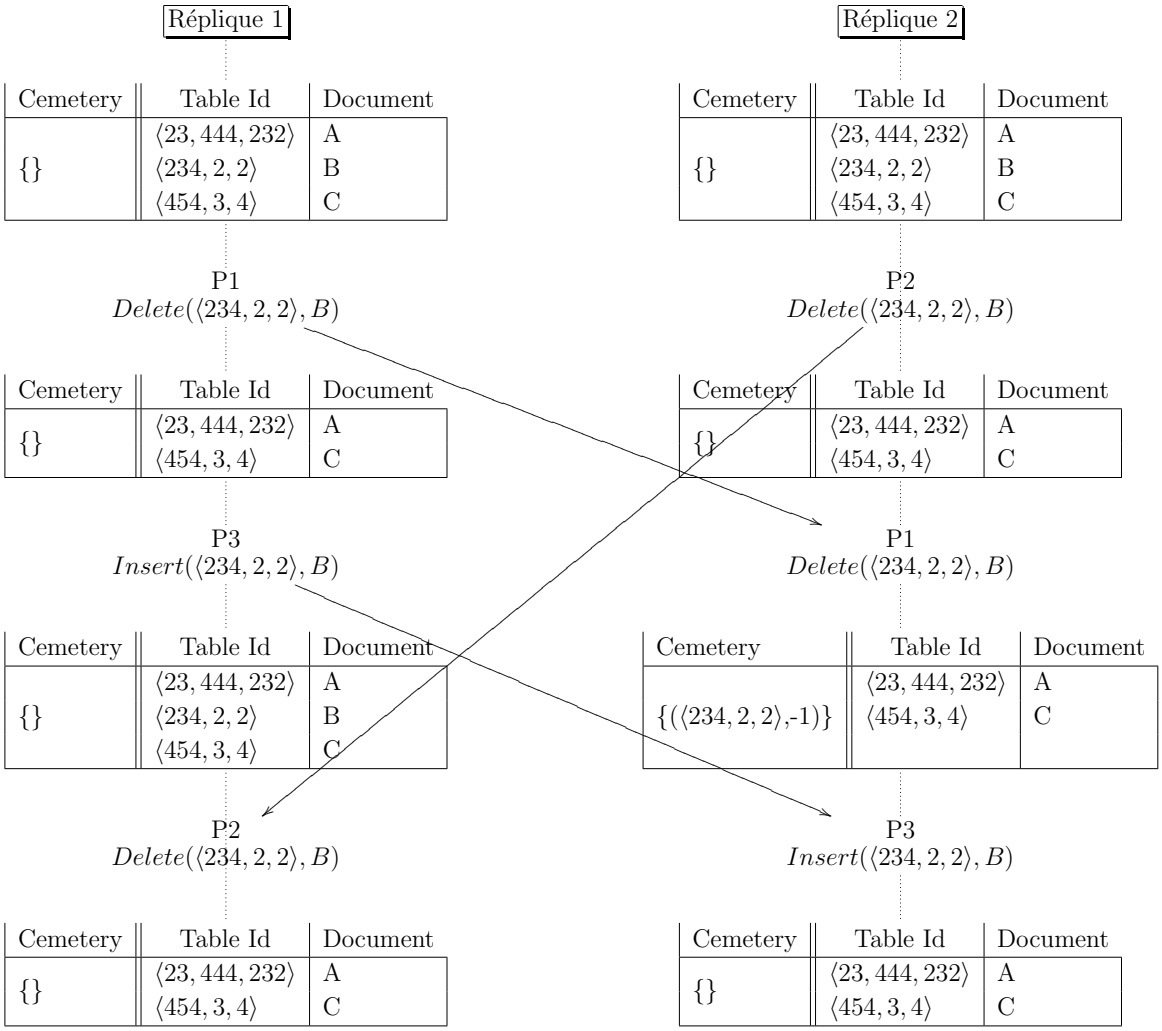


FIGURE 5.7 – Convergence avec les opérations inverses et les degrés.

Cependant dans Logoot$^+$, seule l'exécution du couple d'opérations $(op, Annuler(op, S))$ ne commute pas. En effet, tous les autres couples d'opérations commutent (et pas seulement les couples d'opérations concurrentes) : on peut « supprimer » une ligne (son degré est alors de -1) avant de l' « insérer » (le degré devient alors 0). Afin d'améliorer le passage à l'échelle, on pourrait remplacer la causalité par une précondition sur l'exécution des opérations d'annulation.

Cependant, une réception désordonnée des opérations peut rendre la collaboration difficile. L'utilisation d'un mécanisme de réception causale imparfait qui offre une bonne précision du point de vue de l'utilisateur, tel que les horloges plausibles [TRA99] peut être un compromis pour passer à l'échelle.

5.2.4 Conclusion

Dans ce chapitre, nous avons présenté le modèle $CRDT^+$, une nouvelle approche d'annulation pour les CRDT. Cette approche permet de fournir un mécanisme d'annulation à tout CRDT vérifiant une propriété appelée « inversibilité ». Nous avons présenté les structures de données et les algorithmes propres à notre mécanisme d'annulation.

Nous avons modifié Logoot afin de vérifier la propriété d'inversibilité. Finalement, nous avons combiné Logoot et $CRDT^+$ pour obtenir une nouvelle approche Logoot$^+$ qui permet une édition collaborative avec une fonctionnalité d'annulation vérifiant le modèle de cohérence et compatible avec les caractéristiques des systèmes pair-à-pair pour peu que la longueur des identifiants reste faible.

5.3 Validation expérimentale de Logoot$^+$

Dans cette section, nous vérifions que la fonction d'annulation ne dégrade pas les performances en des proportions inacceptables.

5.3.1 Extension du corpus pour l'annulation

Afin de vérifier l'impact de l'annulation sur la longueur des identifiants, nous étendons le corpus présenté dans le chapitre précédent afin de tenir compte du mécanisme d'annulation. Nous avons extrait les actions d'annulation probables dans les pages considérées. Le seul type d'annulation considéré est la restauration d'une version antérieure. Cette action consiste à annuler toutes les modifications effectuées après cette version. Les restaurations sont détectées en comparant le contenu d'une page wiki avec les 10 dernières révisions de cette page. Si le contenu correspond à une ancienne version, il est probable que l'utilisateur ait utilisé la fonction d'annulation. Les tableaux suivants illustrent le nombre d'annulations pour chaque catégorie du corpus.

Les pages extrêmes

Les pages les plus éditées (table 5.2) contiennent beaucoup d'éditions et le nombre d'annulations dépend fortement de la page considérée. Par exemple, dans la page 1, 18% des éditions sont des annulations alors que sur la page 7, les annulations représentent $0,003\%$ des éditions.

Les articles les plus édités (table 5.3) contiennent un grand nombre d'annulations. En effet, dans les pages considérées, entre 11% et 37% des modifications sont des annulations.

Les articles les plus longs (table 5.4) sont peu édités. Par conséquent, le nombre d'annulations est faible. Le pourcentage d'annulation varie $0,5\%$ pour la page 26 à $17,7\%$ pour la page 21.

id	pages	patches	annulations
1	Wikipedia:Administrator_intervention_against_vandalism	438330	79514
2	Wikipedia:Administrators'_noticeboard/Incidents	343406	4803
3	Wikipedia:Sandbox	170440	25718
4	Wikipedia:Reference_desk/Science	142722	851
5	Wikipedia:Reference_desk/Miscellaneous	148283	1633
6	User:Cyde/List_of_candidates_for_speedy_deletion/Subpage	152974	275
7	Wikipedia:WikiProject_Spam_LinkReports	149538	2
8	Wikipedia:Help_desk	126509	1587
9	Wikipedia:Administrators'_noticeboard	138460	2281
10	Template_talk:Did_you_know	91739	459

TABLE 5.2 – Les pages les plus éditées

5.3.2 Validation

Dans cette section, nous montrons que k, reste petit avec la fonction d'annulation. Nous mesurons la longueur moyenne des identifiants d'une page sur les 100 dernières révisions. Nous présentons aussi la longueur maximale d'un identifiant obtenue sur les 100 dernières révisions.

Les pages les plus éditées La table 5.5 montre la longueur des identifiants en nombre de positions pour les deux stratégies en cas d'annulation.

Pour les pages 2, 4, 5, 8 et 9, les éditions sont principalement des ajouts en fin du document. Des ajouts répétés à la même position représentent le cas défavorable pour notre approche. La stratégie boundary a été proposée pour tolérer les ajouts en fin et donc offre de meilleurs résultats que la stratégie aléatoire, notamment pour les pages 4 et 5. On constate aussi que la fonction d'annulation de Logoot augmente significativement la longueur des identifiants. En effet, elle permet l'annulation des effets du vandalisme, et notamment du vandalisme qui consiste à effacer le contenu de la page. Si on n'utilise pas le mécanisme d'annulation, l'annulation de cette suppression consiste à réinsérer le contenu avec de nouveaux identifiants bien répartis. Dans le cas contraire, le contenu est réinséré avec les identifiants précédents.

Les articles les plus édités Pour les articles les plus édités, on constate que les deux stratégies donnent des résultats très similaires. Ces valeurs sont très proches de la longueur minimale d'un identifiant.

id	pages	patches	annulations
11	George_W._Bush	41563	14449
12	Wikipedia	28977	10953
13	United_States	24781	5770
14	Jesus	20271	4968
15	Wii	19991	4088
16	World_War_II	19547	4373
17	Adolf_Hitler	19489	5978
18	Britney_Spears	19244	3902
19	Deaths_in_2008	19027	2148
20	Michael_Jackson	18956	3735

TABLE 5.3 – Les articles les plus édités

Les articles les plus longs Pour les articles les plus longs, la longueur maximale d'un identifiant de ligne ne dépasse pas 9 positions avec la stratégie aléatoire et 4 positions avec la stratégie « boundary ». En moyenne, la longueur des identifiants de ligne ne dépasse pas 2 positions.

La table 5.8 résume les valeurs moyennes de k pour les pages extrêmes et présente les valeurs pour les pages « choisies aléatoirement ». Hormis pour les pages les plus éditées avec la stratégie aléatoire, k reste petit. On remarque que les pages « choisie aléatoirement », ainsi que les articles les plus édités, obtiennent en moyenne la longueur minimale.

Nous avons donc validé notre approche en vérifiant que la longueur des identifiants reste petite pour un système d'édition collaborative. Par la suite, nous comparons Logoot$^+$ avec les approches existantes.

5.3.3 Comparaison avec les approches existantes

Afin de comparer notre approche, nous avons choisi d'utiliser le corpus d'édition présenté dans la section précédente. Nous nous concentrons uniquement sur la longueur du modèle exprimée en pourcentage de la longueur du document.

Présentation des approches

Nous considérons uniquement les approches respectant le modèle de cohérence CCI, possédant un mécanisme d'annulation et passant à l'échelle sur au moins une caractéristique des systèmes pair-à-pair.

Logoot$^+$ est utilisé dans des conditions similaires à celles de Logoot, c'est-à-dire :
- nous avons ré-exécuté les modifications dix fois pour chaque page afin d'obtenir des valeurs moyennes.

id	page	patches	annulations
21	Line_of_succession_to_the_British_throne	3317	587
22	List_of_World_War_I_flying_aces	640	18
23	Timeline_of_United_States_inventions	3199	129
24	United_States_at_the_2008_Summer_Olympics	2314	50
25	List_of_college_athletic_programs_by_U.S._State	868	6
26	List_of_Brazilian_football_transfers_2008	752	4
27	Licensed_and_localized_editions_of_Monopoly	318	9
28	Austrian_legislative_election,_2008	1086	19
29	List_of_sportspeople_by_nickname	2332	48
30	List_of_mass_murderers_and_spree_killers_by_ number_of_victims	1172	25

TABLE 5.4 – Les articles les plus longs

- une position contient 20 octets,
- nous avons arbitrairement fixé la valeur *boundary* à 1 million.

Wooto associe un identifiant unique par ligne. À cet identifiant s'ajoute un degré pour l'annulation, ce qui implique un coût de 20 octets par identifiant [RWSM+09]. Dans cette approche, les identifiants créés ne sont jamais supprimés, par conséquent, le nombre d'identifiants est égale au nombre d'insertions effectuées sur la page. Par conséquent, le surcoût de Wooto est directement calculé à partir du nombre d'insertions dans la page.

MOT2 est évalué en combinaison avec les fonctions TTF [OUMI06b] étendues pour l'annulation [WUM08]. Ainsi, chaque ligne supprimée est remplacée par un caractère spécial. Ainsi, le surcoût est de 1 octet par ligne supprimée. Ainsi, nous calculons directement le surcoût de MOT2 à partir du nombre de lignes insérées et supprimées.

Comparaison des surcoûts

Les tables 5.9, 5.10 et 5.11 montrent les résultats obtenus en considérant la fonction d'annulation.

La table 5.9 montre un surcoût très important pour la stratégie aléatoire. Cependant, il reste inférieure au surcoût de WOOT et MOT2. Pour les pages 1 à 9, la stratégie boundary possède un surcoût inférieur ou égal à celui de la stratégie aléatoire. Pour la page 10, le surcoût de la stratégie boundary est supérieur à celui de la stratégie aléatoire : les éditions ont lieu majoritairement au début du document, ce qui constitue le pire cas pour la stratégie boundary.

page	Logoot$^+$: stratégie aléatoire		Logoot$^+$: stratégie « boundary »	
	moyenne	maximale	moyenne	maximale
1	1,0	1	1,0	1
2	23,0	28	1,1	2
3	1,0	1	1,0	1
4	1424,0	1433	1,0	2
5	1630,2	1638	1,0	2
6	1,0	1	1,0	2
7	8,8	23	1,0	1
8	1952,9	2051	1,0	1
9	1199,6	1256	1,2	2
10	9,0	11	23,4	32
moyenne	624,2	643,3	1,0	1,5

TABLE 5.5 – Longueur des identifiants de ligne pour les pages les plus éditées en cas d'annulation

page	Logoot$^+$: stratégie aléatoire		Logoot$^+$: stratégie « boundary »	
	moyenne	maximale	moyenne	maximale
11	1,0	1	1,1	2
12	1,0	1	1,0	1
13	1,0	1	1,0	1
14	1,0	1	1,0	2
15	1,0	1	1,0	1
16	1,0	1	1,0	1
17	1,0	1	1,0	1
18	1,0	1	1,1	2
19	1,0	2	1,8	4
20	1,0	1	1,0	1
moyenne	1,0	1,1	1,0	1,6

TABLE 5.6 – Longueur des identifiants de ligne pour les articles les plus édités

page	Logoot$^+$: stratégie aléatoire		Logoot$^+$: stratégie « boundary »	
	moyenne	maximale	moyenne	maximale
21	1,3	3	1,4	4
22	1,0	1	1,8	3
23	1,0	1	1,1	2
24	1,0	2	2,1	4
25	1,0	1	1,0	1
26	3,6	9	1,9	2
27	1,0	1	1,2	2
28	1,3	4	1,7	3
29	1,0	1	1,0	2
30	1,0	2	1,4	3
moyenne	1,3	2,5	1,5	2,6

TABLE 5.7 – Longueur des identifiants de ligne pour les articles les plus longs

Logoot$^+$	pages les plus éditées (pages 1 à 10)	articles les plus édités (pages 11 à 20)	articles les plus longs (pages 21 à 30)	articles de qualité	pages normales
stratégie aléatoire	624.9	1.0	1.3	1.0	1.0
stratégie boundary	3.4	1.0	1.5	1.0	1.0

TABLE 5.8 – Nombre moyen de positions par identifiant de ligne pour les 30 pages considérées.

Pour les tables 5.10 et 5.11, les deux stratégies de Logoot$^+$ donnent exactement le même surcoût. Le surcoût des approches utilisant des pierres tombales est plus important.

5.3.4 Conclusion

Enfin, à partir de l'ensemble de l'expérience, nous déduisons que :

- La stratégie « boundary » de Logoot$^+$ a un surcoût légèrement supérieur à celui de la stratégie « aléatoire » pour les articles les plus longs et les plus édités. Par contre, son surcoût est très fortement inférieur pour les pages les plus éditées. En moyenne, la stratégie « boundary » génère un surcoût largement inférieur.
- Pour les 30 pages considérées, le surcoût moyen en cas d'annulation est :
  - 81838% pour WOOT,
  - 4091% pour MOT2,
  - 2208% pour Logoot$^+$ avec la stratégie « aléatoire »,

page	Logoot$^+$		WOOT	MOT2
	aléatoire	« boundary »		
1	27,8	27,8	211888,6	10593
2	177,9	13,8	989,8	48,9
3	20,2	20,2	1822470,5	91122,5
4	12621,9	8,9	4202,8	209,7
5	16847,3	10,3	7819	390,4
6	27,7	27,8	277643,9	13880,8
7	148,4	11,8	33954,6	1697,1
8	25575,3	13	10426	520,7
9	10270,4	9,8	8765,2	437,8
10	89,2	264,9	1882,4	93,6
moyenne	6580,6	40,8	238004,3	11899,5

TABLE 5.9 – Comparaison du surcoût en mémoire avec annulation pour les pages les plus éditées

– $32,1\%$ pour Logoot$^+$ avec la stratégie « boundary ».

Ces expériences montrent que le surcoût de Logoot$^+$ demeure plus faible que celui des approches existantes.

Les expérimentations que nous avons effectuées n'incluent pas la concurrence. En effet, il n'existe pas de corpus avec des modifications concurrentes. Par conséquent, nous n'avons pas pu mesurer la taille du cimetière dont la taille dépend du nombre d'opérations concurrentes.

page	Logoot$^+$		WOOT	MOT2
	aléatoire	« boundary »		
11	8,3	8,9	18458,6	922,5
12	15,3	15,3	25016,5	1250,1
13	8,3	8,3	5939,3	296,5
14	9,8	9,8	4505,4	224,8
15	8,7	8,7	3524,8	175,8
16	8,3	8,3	5262,3	262,7
17	8,5	8,5	2916,6	145,3
18	9	9,6	3335,5	166,3
19	17,4	33	1635,9	81
20	7,1	7,1	2946	146,9
moyenne	10,1	11,8	7354,1	367,2

TABLE 5.10 – Comparaison du surcoût en mémoire avec annulation pour les articles les plus édités

page	Logoot$^+$		WOOT	MOT2
	aléatoire	« boundary »		
21	30,2	34,2	588,6	28,2
22	71,1	125,3	117,5	2,3
23	10	10,7	100,9	4,6
24	52,8	109,6	351,8	15
25	34,6	34,6	60,7	1,3
26	31,5	16,6	14,1	0,3
27	39,3	45,8	86,3	2,4
28	20,7	25,4	53,7	1,9
29	19,1	19,1	101,9	4,1
30	11,7	15,9	96,3	4,2
moyenne	32,1	43,7	157,2	6,4

TABLE 5.11 – Comparaison du surcoût en mémoire avec annulation pour les articles les plus longs

Chapitre 6

Conclusions et perspectives

6.1 Conclusions

Avec l'arrivée du Web 2.0, l'édition collaborative devient massive. Ce changement d'échelle met à mal les approches existantes qui n'ont pas été conçues pour une telle charge. Afin de répartir la charge, et ainsi, obtenir un plus grand passage à l'échelle, nous nous tournons vers les systèmes pair-à-pair. Ces derniers sont conçus pour passer à l'échelle. Malheureusement, ces systèmes amènent aussi de nouveau défis : dynamicité, la symétrie et bien sur le nombre massif d'utilisateurs et de données. Notre problème est donc de fournir des mécanismes permettant de déployer des systèmes d'édition collaborative de texte sur des environnements pair-à-pair.

Dans ce manuscrit, nous proposons :

- Un *modèle formel* pour les systèmes d'édition collaborative qui nous permet de formaliser le modèle CCI, y compris la préservation de l'intention pour un système d'édition collaborative de texte.
- *Logoot*, un algorithme appartenant à la classe d'algorithme CRDT pour la conception de système d'édition collaborative de texte. L'idée principale de cette approche est d'associer à chaque ligne du document un identifiant unique. L'ensemble de ces identifiants est totalement ordonné et dense, ce qui permet d'exécuter les opérations en temps logarithmique en utilisant l'algorithme de recherche dichotomique. Nous montrons que notre approche vérifie bien le modèle CCI. De plus, nous avons réalisé des expérimentations afin de montrer le passage à l'échelle de notre approche. Cette contribution a été validée par la publication d'un article dans la conférence ICDCS [WUM09].
- *CRDT*$^+$, un type de donnée CRDT dédié à l'annulation. Couplé à un système d'édition collaborative basé sur un CRDT, on obtient un nouveau système d'édition collaborative possédant une fonctionnalité d'annulation. Nous montrons que si le système initial vérifie une propriété que nous proposons, le couple résultant est aussi un CRDT et possède un mécanisme d'annulation est correct.

– *Logoot*$^+$ est la résultante de l'application du CRDT$^+$ sur Logoot. Nous avons modifié Logoot afin qu'il vérifie les propriétés requises. Nous obtenons donc une version de l'algorithme Logoot étendue avec la fonctionnalité d'annulation. De nouvelles expérimentations ont été menées afin de montrer le passage à l'échelle de cette proposition. Cette approche a été publiée dans la revue TPDS [WUM10].

6.2 Perspectives

Dans ce manuscrit, nous avons proposé trois approches « Logoot, CRDT$^+$ et Logoot$^+$ » appartenant à une nouvelle classe d'algorithme appelée CRDT. Les CRDTs sont des algorithmes pour le maintien de la cohérence en environnement réparti. Dans un CRDT, les opérations commutent naturellement, ce qui permet d'obtenir un passage à l'échelle accru. Cette classe d'algorithme est donc tout particulièrement adaptée aux nouveaux systèmes collaboratifs.

Google Wave est un système d'édition collaborative innovant dans lequel les utilisateurs peuvent collaborer en temps réel sur un document hiérarchique représenté par un document XML. Ce système est basé sur l'approche des transformées opérationnelles et utilise l'algorithme d'intégration Jupiter [NCDL95]. Cette solution est centralisée, le serveur ayant la charge de réaliser toutes les transformations. Par conséquent, la collaboration et les performances globales du système reposent uniquement sur le serveur. Afin d'augmenter le passage à l'échelle, la disponibilité du service, il serait intéressant de répartir la charge sur l'ensemble des pairs. Plusieurs approches décentralisées ont été proposées dans le modèle des transformées opérationnelles, cependant, elles reposent sur l'utilisation de vecteurs d'horloge, qui ne sont pas compatibles avec les caractéristiques des réseaux pair-à-pair. Une solution serait donc de proposer un CRDT pour les documents XML. Cela rendrai possible le déploiement d'un système similaire à Google Wave, mais sur un réseau pair-à-pair.

L'informatique dans le nuage (ou Cloud Computing) est un nouveau paradigme à fort potentiel. Buyya et al. définissent l'informatique dans le nuage comme

> « un type de système parallèle et distribué constitué d'une collection d'ordinateurs interconnectés et virtualisés qui est dynamiquement approvisionné et présenté comme une ou plusieurs ressources de calcul unifiée(s), sur la base d'accord de niveau de service établie par négociation entre le fournisseur du service et les consommateurs. »

Par conséquent, il s'agit d'un système appartenant à un fournisseur de service. Ce dernier met à disposition des ressources suivant des accords de niveau de service négociés avec les consommateurs. Le nombre d'ordinateurs physiquement mis à disposition d'un consommateur peut varier au cours du temps. Les applications déployées sur un tel système doivent donc tolérer la dynamicité de ce réseau. De même, les données étant répliquées, l'informatique dans le nuage requiert des algorithmes de maintient de la cohérence. Afin de

passer à l'échelle, le niveau de cohérence généralement appliqué pour cette réplication est la cohérence à terme. Par conséquent, des applications requérant généralement un niveau de cohérence forte doivent être adaptées pour fonctionner avec un niveau de cohérence plus faible. Il est donc nécessaire des proposer des algorithmes de réplication assurant la cohérence à terme pour des types de données nouveaux. Ces algorithmes devant donc passer à l'échelle, les CRDTs sont donc un type de données particulièrement adaptée à ce problème.

Bibliographie

[ABGM90] Rafael Alonso, Daniel Barbará, and Hector Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems (TODS)*, 15(3) :359–384, 1990.

[AD92] Gregory D. Abowd and Alan J. Dix. Giving undo attention. *Interacting with Computers*, 4(3) :317–342, 1992.

[Ama06] Jeff bezos' risky bet, Novembre 2006. `http://www.businessweek.com/magazine/content/06_46/b4009001.htm`.

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4) :335–371, 2004.

[BBM+09] Lamia Benmouffok, Jean-Michel Busca, Joan Manuel Marquès, Marc Shapiro, Pierre Sutra, and Georgios Tsoukalas. Telex : A semantic platform for cooperative application development. In *Conférence Française de Systèmes d'Exploitation (CFSE)*, Toulouse, France, Septembre 2009.

[BCD+99] Andrej Brodnik, Svante Carlsson, Erik D. Demaine, J. Ian Munro, and Robert Sedgewick. Resizable arrays in optimal time and space. In *Workshop on Algorithms and Data Structures*, pages 37–48, Vancouver, British Columbia, Canada, Août 1999.

[Ber90] Brian Berliner. CVS II : Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, Californie, États-Unis, Juin 1990. USENIX Association.

[BJ87] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1) :47–76, 1987.

[BSS91] Kenneth P. Birman, André Schiper, and Pat Stephenson. Lightweigt causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3) :272–314, 1991.

[CD95] Rajiv Choudhary and Prasun Dewan. A general multi-user undo/redo model. In *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work (ECSCW '95)*, pages 231–246, Stockholm, Sweden, Septembre 1995. Kluwer Acadamic.

[CRB+03] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 407–418, Karlsruhe, Germany, Août 2003.

[Cri91] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM (CACM)*, 34(2) :56–78, 1991.

[Cro10] Crowdsourcing. Wikipedia, 2010. `http://fr.wikipedia.org/wiki/Crowdsourcing`.

[CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet : A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, Berkeley, California, Juillet 2000.

[DB08] Bowei Du and Eric A. Brewer. DTwiki : a disconnection and intermittency tolerant wiki. In *WWW '08 : Proceeding of the 17th international conference on World Wide Web*, pages 945–952, Beijing, China, Avril 2008. ACM.

[DGH+87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenkcr, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixteenth annual ACM Symposium on Principles of Distributed Computing - PODC'87*, pages 1–12. ACM Press, Août 1987.

[DGMY03] Neil Daswani, Hector Garcia-Molina, and Beverly Yang. Open problems in data-sharing peer-to-peer systems. Technical Report 2003-1, Stanford InfoLab, Janvier 2003.

[DSU04] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms : Taxonomy and survey. *ACM Computing Surveys*, 36(4) :372–421, 2004.

[EG89] Clarence A. Ellis and Simon J. Gibbs. Concurrency control in groupware systems. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Special Interest Group On Management Of Data (SIGMOD) Conference*, pages 399–407. ACM Press, Mars 1989.

[EGH+03] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4) :341–374, 2003.

[EGR91] Clarence A. Ellis, Simon J. Gibbs, and Gail L. Rein. Groupware : Some issues and experiences. *Communincation of ACM*, 34(1) :39–58, 1991.

[FGR03] Ronaldo A. Ferreira, Christian Grothoff, and Paul Ruth. A transport layer abstraction for peer-to-peer networks. In *In Proceedings of Global and Peer-to-Peer Computing 2003. IEEE Computer Society*, pages 398–405, Tokyo, Japan, Mai 2003.

[Fid91] Colin Fidge. Logical time in distributed computing systems. *Computer*, 24(8) :28–33, Août 1991.

[Fol10] Client statistics by os, Septembre 2010. `http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats`.

[FVC04] Jean Ferrié, Nicolas Vidot, and Michèle Cart. Concurrent undo operations in collaborative environments using operational transformation. In *On the Move to Meaningful Internet Systems 2004 : CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, ODBASE 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 155–173. Springer, Novembre 2004.

[GBRR02] Indranil Gupta, Kenneth P. Birman, Robbert, and Van Renesse. Fighting fire with fire : Using randomized gossip to combat stochastic scalability limits. *Quality and Reliability Engineering International*, 18 :165–184, Mai 2002.

[GP03] Anders Gidenstam and Marina Papatriantafilou. Adaptive plausible clocks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 86–93, Providence, RI, USA, Mai 2003. IEEE.

[HM95] Reza Hazemi and Linda A. Macaulay. User requirements for undo support in cscw. In *British Computer Society Human Computer Interaction*, pages 181–193, 1995.

[IOM+07] Claudia-Lavinia Ignat, Gérald Oster, Pascal Molli, Michelle Cart, Jean Ferrié, Anne-Marie Kermarrec, Pierre Sutra, Marc Shapiro, Lamia Benmouffok, Jean-Michel Busca, and Rachid Guerraoui. A Comparison of

Optimistic Approaches to Collaborative Editing of Wiki Pages. In *Proceedings of the International Conference on Collaborative Computing : Networking, Applications and Worksharing - CollaborateCom 2007*, page 10, White Plains, New York, USA, Novembre 2007. IEEE Computer Society.

[Joh88] Robert Johansen. *GroupWare : Computer Support for Business Teams*. The Free Press, New York, NY, USA, 1988.

[JPPMKA01] Ricardo Jiménez-Peris, Marta Patiño-Martínez, Bettina Kemme, and Gustavo Alonso. How to select a replication protocol according to scalability, availability, and communication overhead. In *Proceedings. 20th IEEE Symposium on Reliable Distributed Systems*, pages 24–33, New Orleans, LA , USA, Octobre 2001.

[JT75] Paul R. Johnson and Robert H. Thomas. RFC 677 : Maintenance of duplicate databases, Janvier 1975. `http://www.faqs.org/rfcs/rfc677.html`.

[KRSD01] Anne-Marie Kermarrec, Antony I. T. Rowstron, Marc Shapiro, and Peter Druschel. The IceCube approach to the reconciliation of divergent replicas. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing - PODC'01*, pages 210–218, Newport, Rhode Island, United States, Août 2001. ACM Press.

[KS98] Ajay D. Kshemkalyani and Mukesh Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing*, 11(2) :91–111, 1998.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communication of the ACM*, 21(7) :558–565, 1978.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2) :133–169, 1998.

[LCL04] Paul Benjamin Lowry, Aaron Curtis, and Michelle René Lowry. Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice. *Journal of Business Communication*, 2004.

[Lin10] Linux. Wikipedia, 2010. `http://en.wikipedia.org/wiki/Linux`.

[LLSG92] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. Providing high availability using lazy replication. *ACM Transactions on Computer Systems (TOCS)*, 10(4) :360–391, 1992.

[LPS09] Mihai Letia, Nuno Preguiça, and Marc Shapiro. CRDTs : Consistency without concurrency control. In *SOSP Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, Big Sky, MT, USA, Octobre 2009.

[Mat89] Friedemann Mattern. Virtual time and global states of distributed systems. In Michel Cosnard et al., editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, Château de Bonas, France, Octobre 1989. Elsevier Science Publishers.

[Med10] MediaWiki. Api :query - properties, 2010. `http://www.mediawiki.org/wiki/API:Query_-_Properties#revisions_.2F_rv`.

[Mic04] Sun Microsystems. Class arraylist<e>, 2004. `http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html`.

[Mic07] Michelle Cart and Jean Ferrié. Asynchronous reconciliation based on operational transformation for p2p collaborative environments. *3rd International Conference on Collaborative Computing : Networking, Applications and Worksharing*, Mai 2007.

[Mor07] Joseph C. Morris. Distriwiki : : a distributed peer-to-peer wiki network. In *Int. Sym. Wikis*, pages 69–74, 2007.

[MR98] Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4) :203–213, 1998.

[MR00] Achour Mostéfaoui and Michel Raynal. Low cost consensus-based atomic broadcast. In *PRDC*, pages 45–52, 2000.

[Mye86] Eugene W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1(2) :251–266, 1986.

[NCDL95] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *UIST '95 : Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 111–120, New York, NY, USA, 1995. ACM.

[NCN88] S. Navaratnam, Samuel T. Chanson, and Gerald W. Neufeld. Reliable group communication in distributed systems. In *International Conference on Distributed Computing Systems*, pages 439–446, 1988.

[Net10] Sourthern Cross Cable Network. Features and benefits, 2010. `http://www.southerncrosscables.com/public/Network/default.cfm?PageID=62`.

[Neu94] B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*, pages 463–489. IEEE Computer Society Press, 1994.

[NR04] Sylvie Noël and Jean-Marc Robert. Empirical study on collaborative writing : What do co-authors do, use, and like ? *Computer Supported Cooperative Work - JCSCW*, 13(1) :63–89, March 2004.

[OB09] Tim O'Reilly and John Battelle. Web squared : Web 2.0 five years on. In *Web 2.0 summit*, 2009.

[OS06] James O'Brien and Marc Shapiro. An application framework for nomadic, collaborative applications. In *Int. Conf. on Dist. App. and Interop. Sys. (DAIS)*, pages 48–63, Bologna, Italy, Juin 2006. IFIP WG 6.1.

[OUM05] Gérald Oster, Pascal Urso, and Pascal Molli. Proving correctness of transformation functions in collaborative editing systems. Rapport de Recherche 5795, INRIA, Décembre 2005.

[OUMI06a] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data Consistency for P2P Collaborative Editing. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*, pages 259–267, Banff, Alberta, Canada, Novembre 2006. ACM Press.

[OUMI06b] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *The Second International Conference on Collaborative Computing : Networking, Applications and Worksharing (CollaborateCom 2006)*, Atlanta, Georgia, USA, Novembre 2006. IEEE Press.

[PK94] Atul Prakash and Michael J. Knister. A framework for undoing actions in collaborative systems. *ACM Transaction on Computer-Human Interaction*, 1(4) :295–330, 1994.

[PMSL09] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *International Conference on Distributed Computing Systems*, Montréal, Québec, Canada, Juin 2009. IEEE Computer Society.

[PRS97] Ravi Prakash, Michel Raynal, and Mukesh Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing*, 41(2) :190–204, 1997.

[PRS07] Stefan Plantikow, Alexander Reinefeld, and Florian Schintke. Transactions for distributed wikis on structured overlays. In *18th IFIP/IEEE International Workshop on Distributed Systems : Operations and Management*, pages 256–267, San José, CA, USA, Octobre 2007.

[PRW05] Ginger Perng, Michael K. Reiter, and Chenxi Wang. Censorship resistance revisited. In *7th international workshop on Information Hiding*, pages 62–76, Barcelona, Spain, Juin 2005.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware IFIP/ACM international conference on distributed systems platforms*, pages 329–350, Heidelberg , ALLEMAGNE, Novembre 2001.

[RG99] Matthias Ressel and Rul Gunzenhäuser. Reducing the problems of group undo. In *International Conference on Supporting Group Work*, pages 131–139, Phoenix, Arizona, USA, Novembre 1999.

[RKL06] Hyun-Gul Roh, Jinsoo Kim, and Joonwon Lee. How to design optimistic operations for peer-to-peer replication. In *Proceedings of the 2006 Joint Conference on Information Sciences, JCIS 2006, Kaohsiung, Taiwan, ROC, Octobre 8-11*, 2006.

[RNRG96] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Computer supported cooperative work*, pages 288–297, 1996.

[Roz08] Simon Rozet. git-wiki : because who needs cool names when you use git ? Wikipedia, 2008.
`http://http://atonie.org/2008/02/git-wiki`.

[RWSM+09] Charbel Rahhal, Stéphane Weiss, Hala Skaf-Molli, Pascal Urso, and Pascal Molli. Undo in peer-to-peer semantic wikis. In *4th Semantic Wiki Workshop (SemWiki 2009) at the 6th European Semantic Web Conference (ESWC 2009)*, Hersonissos, Greece, Juin 2009.

[Sal92] Rich Salz. InterNetNews : Usenet transport for Internet sites. In *USENIX conference proceedings*, pages 93–98, San Antonio, Texas, États-Unis, Été 1992. USENIX.

[SBS07] Pierre Sutra, João Barreto, and Marc Shapiro. Decentralised commitment for optimistic semantic replication. In *International Conference on CO-OPERATIVE INFORMATION SYSTEM*, Vilamoura, Algarve, Portugal, Novembre 2007.

[SCF97] Maher Suleiman, Michèle Cart, and Jean Ferrié. Serialization of concurrent operations in a distributed collaborative environment. In *International Conference on Supporting Group Work*, pages 435–445, 1997.

[SJZ+98] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1) :63–108, Mars 1998.

[SMIRM07a] Hala Skaf-Molli, Claudia-Lavinia Ignat, Charbel Rahhal, and Pascal Molli. New work modes for collaborative writing. In *Enterprise Information Systems and Web Technologies*, pages 176–182, 2007.

[SMIRM07b] Hala Skaf-Molli, Claudia-Lavinia Ignat, Charbel Rahhal, and Pascal Molli. New work modes for collaborative writing. In *International Conference on Enterprise Information Systems and Web Technologies (EISWT-07)*, page 7, Orlando, Florida, USA, Juillet 2007.

[SP07] Marc Shapiro and Nuno Preguiça. Designing a commutative replicated data type. Rapport de recherche INRIA RR-6320, INRIA, Octobre 2007.

[SS05] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1) :42–81, 2005.

[SS06] David Sun and Chengzheng Sun. Operation Context and Context-based Operational Transformation. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*, pages 279–288, Banff, Alberta, Canada, Novembre 2006. ACM Press.

[SS09] David Sun and Chengzheng Sun. Context-based operational transformation in distributed collaborative editing systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(10) :1454–1470, 2009.

[Sta10] Internet World Stats. Internet usage statistics, June 2010. `http://www.internetworldstats.com/stats.htm`.

[Sun00] Chengzheng Sun. Undo any operation at any time in group editors. In *Computer supported cooperative work*, pages 191–200, Philadelphia, Pennsylvania, USA, Décembre 2000.

[SXSC04] David Sun, Steven Xia, Chengzheng Sun, and David Chen. Operational transformation for collaborative word processing. In *Computer supported cooperative work*, pages 437–446, 2004.

[Top10] Top500 list - june 2010, Juin 2010. http://www.top500.org/list/2010/06/100.

[Tor05] Linus Torvalds. git, (April 2005). http ://git.or.cz/.

[TRA99] Francisco J. Torres-Rojas and Mustaque Ahamad. Plausible clocks : Constant size logical clocks for distributed systems. *Distributed Computing*, 12(4) :179–195, 1999.

[TTP+95] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles - SOSP'95*, pages 172–182, Copper Mountain Resort, Colorado, Décembre 1995. ACM Press.

[VCFS00] Nicolas Vidot, Michèle Cart, Jean Ferrié, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Computer supported cooperative work*, pages 171–180, 2000.

[Vid02] Nicolas Vidot. Convergence des copies dans un environnements collaboratifs répartis. Technical report, Université de Montpellier II, septembre 2002.

[Vit84] Jeffrey Scott Vitter. Us&r : A new framework for redoing. *IEEE Software*, 1(4) :39–52, 1984.

[WB84] Gene T. J. Wuu and Arthur J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *ACM Symposium on Principles of Distributed Computing*, pages 233–242, Vancouver, B. C., Canada, Août 1984.

[WEI10] Stéphane WEISS. Mediawikirevisionsextractor, 2010. http://www.sf.net/projects/mediawikire.

[Wik] Wikimedia. Wikipedia. The free encyclopedia that anyone can edit.

[Wik10a] Wikipedia. Wikipedia, 2010. http://en.wikipedia.org/wiki/Wikipedia.

[Wik10b] Wikipedia. Disponibilité. Wikipedia, 2010. http://fr.wikipedia.org/wiki/Disponibilité.

[WUM08] Stéphane Weiss, Pascal Urso, and Pascal Molli. An Undo Framework for P2P Collaborative Editing . In *The 4th International Conference on Collaborative Computing*, Orlando, USA, Novembre 2008.

[WUM09] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot : a Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks . In *International Conference Distributed Computing Systems*, Montréal, Québec, Canada, Juin 2009. IEEE Computer Society.

[WUM10] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot-Undo : Distributed Collaborative Editing System on P2P Networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8) :1162–1174, 2010.

[YV01] Haifeng Yu and Amin Vahdat. The costs and limits of availability for replicated services. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 29–42, Banff, Alberta, Canada, Octobre 2001.

Résumé

Avec l'arrivée du Web 2.0, l'édition collaborative devient massive. Ce changement d'échelle met à mal les approches existantes qui n'ont pas été conçues pour une telle charge. Afin de répartir la charge, et ainsi, obtenir un plus grand passage à l'échelle, de nombreux systèmes utilisent une architecture dite pair-à-pair. Dans ces systèmes, les données sont répliquées sur plusieurs pairs et il est alors nécessaire de définir des algorithmes de réplication optimiste adaptés aux caractéristiques des réseaux pair-à-pair : la dynamicité, la symétrie et bien sûr le nombre massif d'utilisateurs et de données. De plus, ces systèmes étant des éditeurs collaboratifs, ils doivent vérifier le modèle de cohérence dit « CCI » (Causalité, Convergence et Intention).

Dans ce manuscrit, nous proposons un *modèle formel* pour les systèmes d'édition collaborative qui nous permet de formaliser le modèle CCI. Dans ce modèle, nous proposons *Logoot*, un type de données répliqué commutatif (CRDT) pour les documents texte. Par la suite, nous définissons un mécanisme d'annulation générique pour les types de données *CRDT*. Nous appliquons notre mécanisme d'annulation sur *Logoot* pour obtenir un CRDT texte avec la fonctionnalité d'annulation appelée *Logoot*$^+$. Nous proposons finalement une évaluation comparative des approches *Logoot* et *Logoot*$^+$ à partir des modifications produites sur plus de 2000 pages de Wikipédia.

Mots-clés: Travail collaboratif, Réseau Poste-à-Poste, Traitement réparti

Abstract

With the arrival of Web 2.0, collaborative editing becomes massive. This scale change is undermining the existing approaches that were not designed for such a charge. To distribute the load, and thereby achieve greater scalability, many systems use an architecture known as peer-to-peer. In these systems, data is replicated on several peers and it is then necessary to define optimistic replication algorithms adapted to the characteristics of peer-to-peer : dynamicity, symmetry and of course the massive number of users and data. Moreover, these systems are collaborative editors, they should check the model of consistency called "CCI" (Causality, Convergence and Intention).

In this manuscript, we propose a *formal model* for collaborative editing systems that enables us to formalize the CCI model. In this model, we propose *Logoot* a commutative replicated data type (CRDT) for text documents. Subsequently, we define an undo mechanism for generic *CRDT*. We apply our undo mechanism on *Logoot* to get a CRDT text with the undo feature called *Logoot*$^+$. We finally propose a comparative evaluation of approaches *Logoot* and *Logoot*$^+$ from the changes produced over 2000 pages of Wikipedia.

Keywords: Collaborative Editing, Peer-to-Peer, Distributed computing