Évaluation de l'occupation mémoire des CRDTs pour l'édition collaborative temps-réel mobile

Mehdi Ahmed-Nacer, Pascal Urso, Claudia-Lavinia Ignat, Gérald Oster

Évaluation de l'occupation mémoire des CRDTs pour l'édition collaborative temps-réel mobile [1]

Mehdi Ahmed-Nacer [1, 2, 3], Pascal Urso [1, 2, 3], Claudia-Lavinia Ignat [3, 1, 2], Gérald Oster [1, 2, 3]
[1] Université de Lorraine, LORIA, Nancy, F-54000, France
[2] CNRS, LORIA, Vandoeuvre, F-54500, France
[3] Inria, Villers-lès-Nancy, F-54600, France
{mahmedna, urso, ignatcla, oster}@loria.fr

Résumé :

Les systèmes d'édition collaborative temps réel connaissent un large succès depuis quelques années. Ils permettent à plusieurs utilisateurs de collaborer pour rédiger un même document. Les appareils mobiles sont omniprésents dans notre vie quotidienne, et, par conséquent, le besoin d'applications dédiées à l'édition collaborative devient naturel.

Dans ce cadre d'édition collaborative, chaque utilisateur à sa propre copie du document sur son dispositif mobile. Un mécanisme de réplication est donc nécessaire pour supporter l'édition concomitante et assurer la disponibilité des données tout en préservant la cohérence des copies. Récemment, une nouvelle classe de mécanismes de réplication dénommée CRDTs (Commutative Replicated Data Types), a été introduit pour l'édition collaborative. Cependant, leur pertinence face aux contraintes de ressources des appareils mobiles telles que les performances, la batterie et la mémoire limitée reste à étudier.

Cet article présente un travail en cours sur l'évaluation des CRDTs pour l'édition collaborative en temps réel. Il présente une première évaluation des performances en termes de temps de calcul et d'occupation en mémoire pour les différents CRDTs. Il fournit également une comparaison avec un mécanisme de réplication traditionnel.

Mots clés : *Collaboration, Performance, Mobilité, Réplication, CRDT.*

1 Introduction

Il y a quelques années de cela, les principaux soucis des utilisateurs consistaient seulement à trouver des informations statiques sur Internet. Aujourd'hui, Internet est plus collaboratif, il facilite l'interaction entre les utilisateurs et permet aux internautes de contribuer, d'interagir et de partager les informations de façon simple, rapide et en temps réel.

Récemment, la collaboration en temps réel est devenue un des enjeux majeurs d'Internet, elle permet à plusieurs utilisateurs situés à des endroits différents d'éditer un document au même moment par le biais d'outils tels que GoogleDocs ou Microsoft SharePoint. Le caractère temps-réel de ces applications est lié à la contrainte de temps de propagation d'une action utilisateur, i.e. chaque modification effectuée par un utilisateur doit être très rapidement visible pour les autres utilisateurs qui participent à l'édition.

Les dispositifs mobiles tels que les ordinateurs portables, les tablettes, netbook et les téléphones cellulaires sont de plus en plus utilisés de nos jours. Ils permettent aux utilisateurs de travailler efficacement dans divers endroits, loin de leurs bureaux. Supporter l'édition collaborative sur des appareils mobiles devient un besoin réel et la progression de la technologie mobile a conduit au développement d'applications d'éditions collaboratives telles que NetSketch sur iPhone ou encore GoogleDocs sur les appareils Android. Pour que les appareils mobiles soient plus efficaces pour échanger des informations entre les utilisateurs et qu'il supportent le travail mobile, la conception d'applications d'édition collaborative doit soigneusement prendre en compte les

[1] Ce travail est partiellement financé par les programmes nationaux de recherche français ConcoRDanT (ANR-10-BLAN-0208) et STREAMS (ANR-10-SEGI-010).

contraintes de ces systèmes: la mémoire limitée, la puissance de traitement limitée, une bande passante limitée, l'autonomie de la batterie, etc.

Dans d'édition collaborative en temps réel, pour des raisons de disponibilité et de performance, les données sont dupliquées. Chaque copie est une copie locale du document et quand un utilisateur génère une action, cette dernière est transformée en un ensemble d'opérations qui sont exécutées localement et envoyées aux autres copies. Lorsqu'une copie reçoit une opération distante, elle l'exécute localement. Pour maintenir la cohérence du document partagé, une approche dite des transformées opérationnelles (OT) a été proposée comme un mécanisme approprié pour la collaboration en temps réel [2, 3, 7, 8, 9, 10, 11]. Elle assure la cohérence des données en adaptant les paramètres des opérations distantes afin de prendre en compte les effets des opérations concomitantes. Par exemple, GoogleDocs repose sur un algorithme OT dénommé Jupiter [8].

Récemment, une nouvelle approche dite Commutative Replicated Data Types (CRDT) [12, 13, 14, 15] a été proposée comme substitut de l'approche OT pour assurer la cohérence des données dans les réseaux pair-à-pair. Contrairement à l'approche des transformées opérationnelle, les CRDTs n'utilise pas l'historique des opérations pour détecter la concomitance entres les opérations afin d'assurer la cohérence. L'approche CRDT a été conçue en se basant sur des opérations concomitantes nativement commutatives.

Cependant, la pertinence des approches OT et CRDT sous contraintes de ressources mobiles telles que la puissance de calcul et la mémoire limitée reste à étudier. Comme la plupart des appareils mobiles ont une mémoire de stockage limitée, l'espace mémoire requis par ces algorithmes est une contrainte cruciale. Le temps de calcul consommé par ces algorithmes doit également être le plus faible possible afin d'économiser la batterie et d'améliorer la réactivité des applications. Dans cet article, nous allons évaluer la performance des algorithmes OT et CRDT en terme d'occupation mémoire et de temps de calcul dans le cadre de l'édition collaborative en temps réel.

Ce document est structuré comme suit. Nous commençons par présenter un aperçu des principaux algorithmes évalués dans notre expérience. Ensuite nous donnons une brève description de notre expérience pour collecter des traces d'édition collaboration en temps réel. Nous fournissons ensuite les résultats d'évaluation en mémoire pour les algorithmes CRDT et OT sur des traces réelles. Dans la dernière section nous concluons notre travail avec quelques remarques et les travaux en perspective.

2 Évaluation des algorithmes

Dans les sections suivantes, nous décrivons brièvement les algorithmes CRDT et OT que nous avons évalués et nous précisons pour chaque algorithme leurs avantages et inconvénients pour des applications mobiles. Tous les algorithmes évalués sont conçus pour l'édition collaborative d'un document dont la structure est linéaire (le document est représenté comme une séquence de caractères). Les opérations qui peuvent être exécutées sur le document sont l'insertion et la suppression de caractères.

2.1 WOOT

WOOT [12] est le premier algorithme CRDT qui a été proposé. L'idée de base de WOOT est d'identifier chaque élément par une structure unique. La structure est définie en spécifiant l'identifiant du nouveau élément, le contenu de l'élément et les identifiants qui précédent et suivent cet élément.

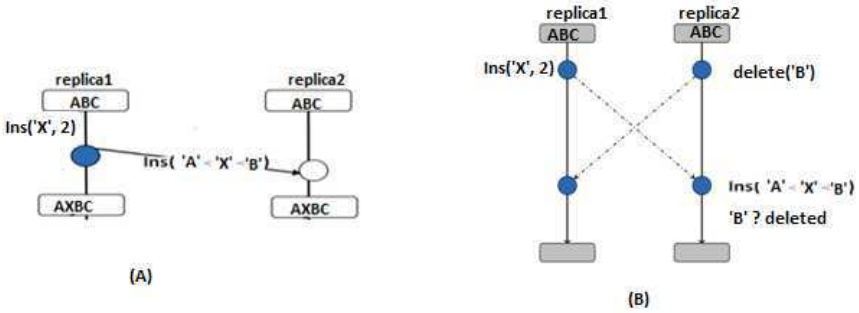

FIG. 1 – *Exécution d'une insertion et d'une suppression dans WOOT.*

La Figure 1 présente l'exécution d'une insertion dans WOOT et illustre le problème de la suppression dans cet algorithme. Dans la Figure 1(a), l'utilisateur 1 insère le caractère 'X' à la position 2 et envoie cette opération à un autre utilisateur. L'opération envoyée contient l'identifiant du caractère 'X' et les identifiants qui précèdent et suivent le caractère 'X' lors de son insertion : 'A' et 'B'. WOOT utilise un ordre global pour ordonner les caractères insérés de manière concomitante à la même position.

Un des problèmes de WOOT est que l'algorithme doit toujours pouvoir retrouver les éléments suivant et précédent, comme illustré à la Figure 1(b). Pour éviter ce problème, WOOT conserve tous les éléments supprimés (appelés pierres tombales) dans le document. Les éléments ne sont pas supprimés mais seulement rendus invisibles aux utilisateurs. Ceci représente un inconvénient majeur, car le document ne pourra que grossir durant l'édition collaborative, ce qui pourrait devenir un problème face à la capacité mémoire limitée d'un périphérique mobile.

D'un autre côté, WOOT est adapté pour les groupes d'utilisateurs ouverts où les utilisateurs peuvent rejoindre et quitter le réseau. En outre, l'algorithme ne nécessite pas de mécanisme de respect de la causalité des opérations et les opérations sont intégrées dans un ordre quelconque sur chaque copie. Deux solutions optimisées de WOOT ont été proposées: WOOTO [16] et WOOTH.

WOOTO [16] est une optimisation de WOOT qui améliore sa complexité algorithmique, il introduit la notion de *degré* pour comparer les éléments non ordonnés. WOOTH est une version améliorée de WOOTO. Il utilise une table de hachage et une liste chaînée afin d'optimiser la récupération et l'insertion d'éléments dans le document. WOOTO et WOOTH sont basés sur le même principe, garder les éléments supprimés et mettent en péril la contrainte de mémoire limitée. Dans notre expérience, nous nous sommes limités à l'évaluation uniquement des versions optimisées de WOOT, c.à.d. WOOTO et WOOTH.

2.2 Logoot

Logoot [15] est une autre approche CRDT pour l'édition de documents texte. Comme WOOT, Logoot associe un identifiant unique à chaque élément dans le document. L'identifiant de Logoot est représenté par une liste de triplet. Chaque triplet <position, site-id, horloge> contient une position d'insertion (une valeur entière), un identifiant de copie et une horloge logique (le compteur du nombre de modifications générées par la copie).

Contrairement à WOOT, les éléments supprimés dans Logoot sont physiquement retirés, la notion de pierre tombale n'existe plus. Cela peut représenter un avantage majeur pour les appareils mobiles pour économiser de l'espace mémoire. Toutefois, l'inconvénient de Logoot est la taille des identifiants qui peuvent grandir sans limite et nécessiter un espace mémoire important.

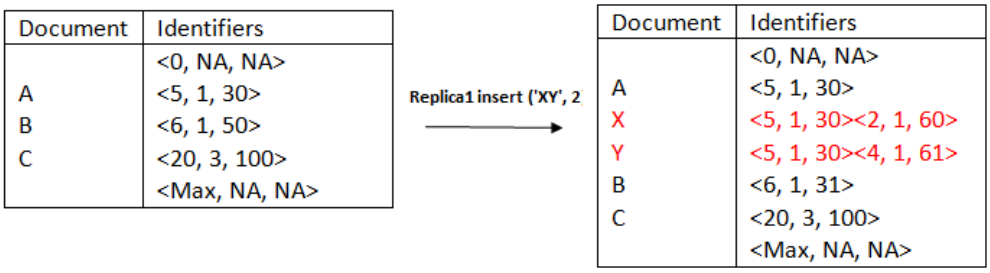

FIG. 2 – *Insertion dans Logoot.*

Dans la Figure 2, l'utilisateur souhaite insérer deux caractères 'X' et 'Y' entre les caractères 'A' et 'B' dont les deux identifiants uniques sont <5, 1, 30> et <6, 1, 50>. Le nombre de places disponibles entre ces deux identifiants est 0. De ce fait, pour insérer 'X' et 'Y', il est nécessaire de créer des identifiants dont la longueur est 2.

2.3 RGA

Comme Logoot et WOOT, RGA (Replicated Growable Array) [14] est un algorithme CRDT conçu pour l'édition collaborative d'une structure linéaire. RGA maintient une liste chainée d'éléments. Une opération locale trouve sa cible en utilisant un indice entier. Une table de hachage est utilisée par une opération distante pour extraire l'objectif par un indice unique. Un indice unique dans RGA est représenté par une structure dénommée s4vector [14] comprenant quatre nombres entiers.

Le vecteur s4vector est délivré à chaque opération, il peut être utilisé comme l'indice pour trouver une cible dans la table de hachage ; il est également utilisé pour résoudre les conflits entre les insertions concomitantes à la même position.

Une insertion compare le s4vector qui lui est associé avec les identifiants s4vector des éléments proches de son élément cible, et ajoute son nouvel élément devant le premier élément rencontré qui possède le s4vector le plus ancien. L'insertion la plus récente insère l'élément à droite de sa position cible avec une priorité plus élevée que les insertions concomitantes. Cette transitivité de priorité, réalisée avec s4vectors, assure la cohérence des insertions concomitantes. Comme WOOT, RGA utilise également des pierres tombales pour les éléments supprimés.

2.4 SOCT2

SOCT2 [9] est un algorithme qui repose sur l'approche des transformées opérationnelles pour assurer la cohérence des copies. Il maintient un vecteur d'horloge pour assurer la causalité. Lorsque l'utilisateur génère une opération, celle-ci est immédiatement exécutée localement puis envoyée à toutes les autres copies, y compris lui-même. Cela a pour effet d'ajouter cette opération dans l'histoire locale. L'opération est diffusée vers les autres copies, sous la forme d'un vecteur avec trois paramètres: type de l'opération, l'identifiant du site qui a généré l'opération et le vecteur d'horloge de l'opération. Le principe de cet algorithme est que quand une opération distante doit être intégrée, tout l'historique des opérations déjà exécutées est traversé et réordonné afin de déterminer qu'elles sont les opérations concomitantes. Puis, l'opération distante est transformée par rapport à ces opérations (mécanisme de *transposition en avant)* afin d'obtenir une forme de l'opération qui soit exécutable sur l'état courant, c.à.d. qui tienne compte de l'exécution précédente des opérations de l'historique qui lui sont concomitantes.

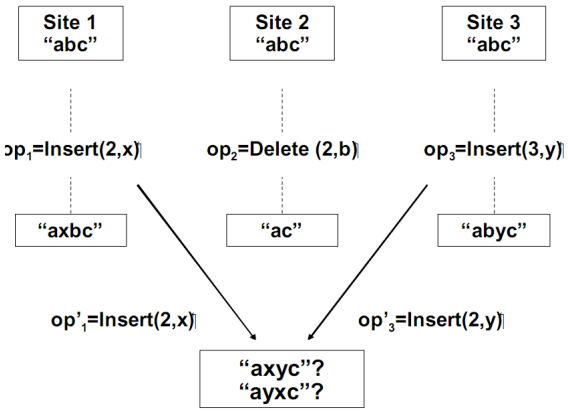

FIG. 3 – *Problème de divergence.*

Les propriétés C1 et C2 [3, 9] assurent que la transformation de toute opération par rapport à une séquence d’opération concomitante dans différents ordres d'exécution donne toujours le même résultat. Ces propriétés garantissent la convergence des copies quelque soit l’ordre d’exécution des opérations concomitantes (cf. Figure 3). Malheureusement, de nombreuses fonctions de transformation proposées ne satisfont pas ces conditions [17]. Les seules fonctions de transformation existantes qui répondent aux conditions C1 et C2 sont celles proposées par l’approche TTF (Tombstone Transformation Functions) [18]. Pour surmonter les problèmes, l'approche TTF conserve tous les caractères dans le modèle du document, les caractères supprimés sont remplacés par des pierres tombales.

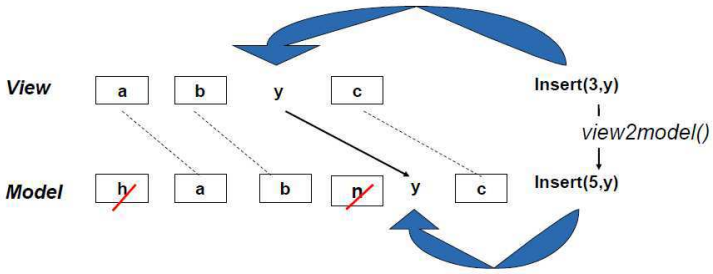

FIG. 4 – *Modèle dans l’approche TTF.*

Théoriquement, cette approche introduit une surcharge en occupation mémoire en raison de la présence de pierres tombales et au stockage des opérations dans l'historique. De plus, le mécanisme de *transposition en avant* consomme beaucoup des ressources de calcul puisque l’algorithme réordonne l’historique à chaque fois qu’une opération distante doit être intégrée. Par conséquent, la mémoire limitée et la puissance de traitement limitée d'un appareil mobile pourraient faire que cette approche est inadaptée aux appareils mobiles.

2.5 Analyse des algorithmes

Lors de la conception d’applications d'édition collaborative pour les appareils mobiles, deux contraintes majeures doivent être prises en compte : d'un côté la puissance de traitement limitée et la consommation de batterie et de l'autre côté la mémoire limitée des périphériques mobiles par rapport aux ordinateurs de bureau.

Dans les prochaines sections, nous allons évaluer les algorithmes présentés en termes de temps de calcul et d'espace mémoire nécessaire. Nous fournissons à la fois une évaluation théorique et une évaluation pratique basée sur des traces réelles de collaboration.

2.6 Évaluation théorique

Dans cette section, nous fournissons une évaluation théorique des algorithmes étudiés. La complexité dans le pire des cas et en moyenne pour chacun des algorithmes décrits ci-dessus est présentée dans le Tableau 1.

Nous désignons par:

- *R* le nombre de copies
- *H* le nombre d'opérations qui ont affecté le document
- *c* le nombre moyen d'opérations concomitantes
- *n* la taille du document (sans caractères supprimés)
- *N* le nombre total de caractères insérés (y compris ceux qui ont été supprimés)
- *k* la taille moyenne des identifiant Logoot
- t = N - n, la taille des pierres tombales
- d = [(t + c) / n] le nombre d'éléments entre deux éléments successifs.

ALGORITHMES	COMPLEXITÉ EN ESPACE	
	PIRE CAS	MOYENNE
WOOT-WOOTO-WOOTH	*O(H)*	*O(N)*
Logoot	*O(H²)*	*O(k.n)*
RGA	*O(H)*	*O(N)*
SOCT2/TTF	*O(H.R)*	*O(H.R)*

TAB 1 – *Analyse de la complexité en espace.*

En moyenne, les algorithmes basés sur les pierres tombales ont besoin de stocker *N* éléments dans leur modèle. Logoot stocke *n* identifiants avec une taille moyenne de *O(k)*. SOCT2 stocke de plus un historique des opérations, chacune contenant un vecteur dont la taille est de *O(R)*.

ALGORITHMES	EXÉCUTION LOCALE		EXÉCUTION DISTANTE	
	INS	DEL	INS	DEL
Logoot	*O(k)*	*O(1)*	*O(k.log(n))*	*O(k.log(n))*
WOOTO	*O(N.d²)*	*O(N)*	*O(N+d²)*	*O(N)*
WOOTH	*O(N+d²)*	*O(N)*	*O(d²)*	*O(1)*
RGA	*O(N)*	*O(N)*	*O(1+c/n)*	*O(1)*
SOCT2	*O(N+R)*	*O(N+R)*	*O(H.c)*	*O(H.c)*

TAB 2 – *Analyse de la complexité algorithmique moyenne.*

Les algorithmes basés sur les pierres tombales (WOOTs, RGA et SOCT2) ont une complexité en fonction de N, ceci est dû à la récupération d'un élément ou d'une position dans leur modèle. Contrairement à Logoot, qui à une complexité de *O(k)* lors de la création de l'identifiant localement et *O(1)* pour la suppression d'un élément avec l'identifiant unique. Lors de la récupération d'une opération distante, RGA ne compare l'élément qu'avec une autre opération concomitante insérée à la même place (c / n en moyenne). L'algorithme SOCT2 réordonne l'historique à chaque fois qu'une opération est reçue avec en moyenne *O(c)* opérations concomitantes.

3 Évaluation expérimentale

Dans cette section, nous présentons notre évaluation expérimentale.

3.1 Évaluation expérimentale sur des traces réelles

Dans [19] nous avons effectué quelques études avec des utilisateurs sur une session d'édition collaborative pair-à-pair en temps-réel et nous avons recueilli des traces réelles pour ce type de collaboration. Nous avons modifié le code source d'un logiciel d'édition temps réel de documents appelé TeamEdit[2] pour récolter les données de notre expérience. Nous avons demandé à des étudiants de rédiger des documents en collaboration en utilisant TeamEdit et enregistré un certain nombre d'opérations générées au cours de cette expérience. Dans la section 3.2 nous allons donner une brève description de l'expérience que nous avons effectuée. Dans cet article, nous exécutons les algorithmes étudiés sur les mêmes traces recueillies dans [19], mais cette fois, nous évaluons l'espace mémoire occupé par chaque algorithme.

Tous les algorithmes décrits précédemment sont mis en œuvre dans un framework appelé *ReplicationBenchmark* développé en Java, et disponible sur la plateforme GitHub[3] sous la licence libre GPL. Le framework fournit des classes de base communes pour les différents algorithmes d'édition collaborative en temps réel, tels que le document, la structure des vecteurs d'horloges, les opérations, etc. Ainsi chacun de ces algorithmes peut être implémenté par des classes dérivées.

Le Framework demande à chaque algorithme de générer des opérations dans son propre format à partir des traces fournies dans le journal à rejouer. Pour obtenir un résultat correct de la taille mémoire occupée par chaque algorithme, nous sérialisons chaque document d'une copie à l'aide du mécanisme de *sérialisation* de Java après chaque groupe de dix opérations.

Dans un premier temps nous exécutons chaque algorithme cinq fois sur des traces réelles. Toutes les exécutions ont eu lieu sur la même JVM, sur une machine bi-processeur Intel® Xeon® 5160 à double-coeurs (4Mb Cache, 3.00 GHz, 1333 MHz FSB) sous GNU/Linux 2.6.9-5. Pendant l'expérience, un seul coeur est utilisé pour la mesure, et après chaque exécution, nous exécutons le ramasse miette de Java.

3.2 Comportement des algorithmes sur des traces réelles

Afin de collecter des traces réelles à partir d'une édition collaboration en temps-réel, nous avons effectué deux expériences de collaboration avec des groupes d'étudiants « rapport » et « séries ». La première collaboration (rapport) a été réalisée avec des groupes de 4 étudiants de Master. Nous leur avons demandé d'écrire en collaboration un rapport de leur projet de semestre pour le cours de génie logiciel sous TeamEdit, sans l'aide d'autres outils de communication. Dans la seconde collaboration (série) qui a duré environ une heure et demie, nous avons demandé à 18 étudiants de regarder un épisode de la série "The Big Bang Theory" et de produire une transcription de l'épisode tout en la regardant. Dans le tableau 3, nous représentons certaines caractéristiques des traces réelles collectées dans notre expérience.

LES DONNÉES	RAPPORT	SÉRIE
NOMBRE D'OPERATIONS	11211	9042
NOMBRE DE REPLICAS	4	18
% D'INSERTION	88%	91%
% DE COPIER-COLLER OU DE SUPPRESSION DE BLOC DE CARACTERES	2.39%	4.04%

TAB 3 – *Caractéristique des traces réelles.*

[2] TeamEdit. *A collaborative text editor*. http://teamedit.sourceforge.net/.
[3] http://github.com/PascalUrso/ReplicationBenchmark

Afin d'obtenir des représentations significatives pour le comportement des algorithmes, nous sérialisons chaque copie après avoir joué dix opérations et enfin nous calculons la moyenne de la taille des copies sérialisées. Nous présentons dans ce qui suit, l'occupation de mémoire pour algorithmes dans la première expérimentation.

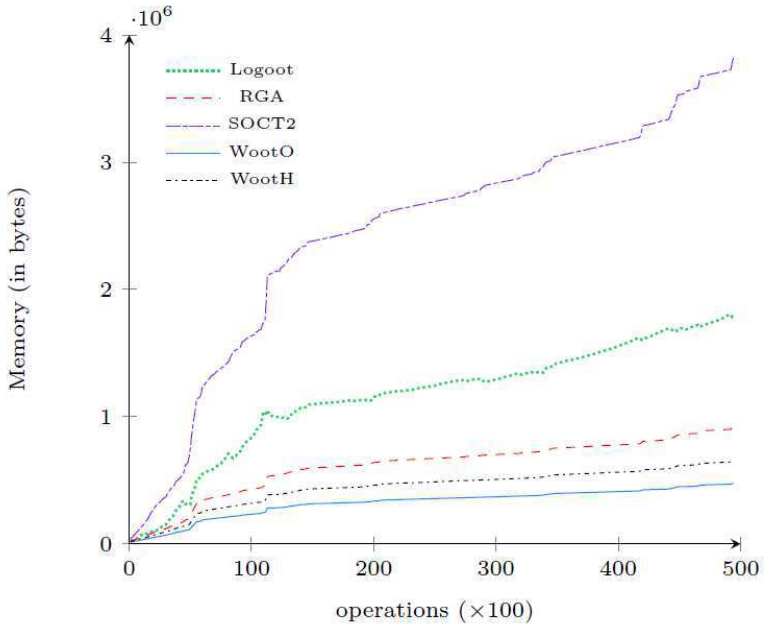

FIG. 5 – *Occupation mémoire par les algorithmes dans la première collaboration (rapport).*

Au début de l'expérience, entre la première opération et la 10000ièmes opérations, la mémoire occupée augmente rapidement, spécialement pour Logoot et SOCT2. Pour Logoot, cela s'explique par la taille des identifiants qui connaît une croissance rapide par rapport aux autres algorithmes qui sont basés sur des pierres tombales comme RGA, WOOTO et WOOTH. Cela est dû aux copier/coller au début de l'expérience, car les étudiants ont commencé par récupérer des travaux existants. SOCT2 consomme le plus de mémoire car il garde toutes les opérations dans l'historique. Comme nous l'avons montré dans le Tableau 1, la complexité est en *O(H.R)*, ce qui explique son comportement linéaire. Les comportements globaux de RGA et de la famille WOOT sont assez similaires (en particulier pour les WOOTO et WOOTH), même si ce sont des algorithmes très différents. Pour ces algorithmes, la principale cause de la croissance de l'occupation mémoire est la présence des pierres tombales. Logoot est le moins efficace de toutes les approches CRDTs en terme d'occupation mémoire tandis que SOCT2 est le pire parmi toutes les approches.

Le comportement des algorithmes dans la deuxième expérimentation (série) est similaire à la première collaboration (rapport) en dépit du fait que certaines caractéristiques sont différentes comme le nombre d'opérations simultanées et le pourcentage de suppressions.
Il faut noter que nous faisons un appel périodique au mécanisme de ramasse miettes de Java pour purger de la mémoire les objets non référencés.

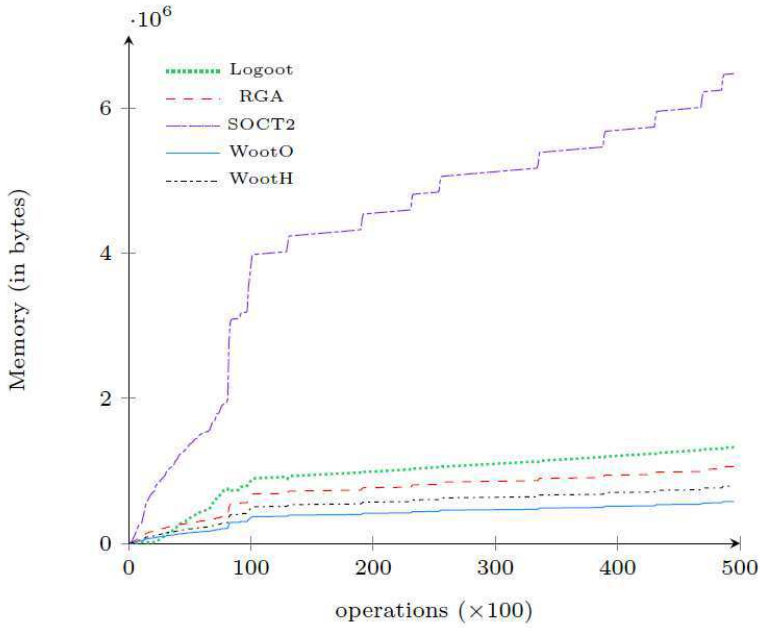

FIG. 6 - *Occupation mémoire par les algorithmes dans la seconde collaboration (série).*

Afin d'évaluer les algorithmes en temps de calcul et observer le temps d'occupation de CPU, nous avons effectué une étude sur le temps d'exécution des algorithmes sur les deux expériences. Le Tableau 4 présente le temps d'exécution moyen d'une séquence d'opérations pour chaque algorithme. Une étude approfondie sur la performance des algorithmes, notamment les temps moyen et maximum d'exécution d'une opération ont été réalisés dans [19].

	TEMPS MOYEN (*ms*)	
	RAPPORT	SÉRIE
Logoot	102 772	67 272
WOOTO	737 327	1267 505
WOOTH	54 110	241 921
RGA	44 466	22 710
SOCT2	62 680 857	101 443 082

TAB 4 – *Temps d'exécution moyen pour chaque algorithme en (ms).*

SOCT2 est le moins adapté des algorithmes pour les appareils mobiles en terme de temps de calcul. En effet, le temps d'exécution avec SOCT2 pour la première expérimentation prend en moyenne plus d'une minute. Tandis que les algorithmes basés sur une table de hachage ne dépassent pas une seconde. WOOTO qui est le meilleur en terme d'occupation mémoire est moins performant en temps de calcul. Logoot qui est le moins performant en mémoire par rapport aux autres algorithmes CRDTs, n'est pas meilleur en temps d'exécution. Le meilleur algorithme en temps d'exécution est RGA avec seulement 44 *ms* dans la première collaboration et 22 *ms* en deuxième.

D'autre part les appareils mobiles ont une capacité de traitement limitée et une contrainte de consommation de la batterie. Un bon algorithme pour les appareils mobiles doit avoir une complexité algorithmique faible. Comme nous pouvons le remarquer dans le Tableau 4, SOCT2 possède le pire temps d'exécution qui se traduit par la consommation la plus élevée du processeur.

WOOTO est l'algorithme le plus adapté pour les applications mobile pour l'édition collaborative en temps réel en terme d'occupation mémoire, tant dis que RGA est moins exigent en CPU.

4 État de l’art

La première évaluation d'algorithmes OT a été présentée dans [21]. Toutefois, ce travail n'a pris en compte ni les approches CRDT ni l'évaluation des algorithmes en terme d’occupation mémoire. Dans [20] des algorithmes OT pour des connexions intermittentes dans un contexte mobile ont été proposés et évalués. Cependant, aucune comparaison n'a été faite avec des approches CRDT et aucune évaluation en espace mémoire n’a été réalisée. L’occupation mémoire de Logoot a déjà été évaluée à l'aide des traces extraites de Wikipedia. Cependant, ces traces contiennent une sérialisation des opérations concomitantes. Elles ne peuvent donc pas être utilisées pour l'évaluation d’algorithmes pour d’édition collaboration en temps réel. Les algorithmes WOOT et RGA n'ont jamais été évalués sur leurs besoins en mémoire. Dans [19], nous avons présenté pour la première fois une comparaison des temps d'exécution pour tous les algorithmes ci-dessus sur des traces réelles, mais nous n'avons pas évalué l'espace mémoire requis par ces algorithmes. À cet égard, ce document est le premier qui présente la comparaison en terme d’occupation d’espace mémoire requis pour les algorithmes OT et CRDTs dans une édition collaborative en temps réel. Tous les algorithmes ont été écrits dans la même langage et évalués dans le même environnement. En outre, ceci est le premier travail qui évalue les algorithmes en utilisant des traces de collaboration qui ont été générées au cours de véritable sessions d'édition collaborative en temps-réel.

5 Conclusion et perspectives

Dans ce papier, nous avons évalué des algorithmes CRDT et OT qui peuvent être utilisés sur des appareils mobiles pour des applications d’édition collaborative en temps réel en tenant compte de leurs besoins en espace mémoire. Nous avons fourni une évaluation théorique ainsi qu'une étude expérimentale sur des traces réelles de collaboration en temps-réel. Nous avons découvert que les algorithmes CRDT imaginés pour des réseaux pair-à-pair sont adaptés pour les applications d’édition collaborative en temps réel sur des appareils mobiles. En outre, elles surpassent certaines approches représentatives du modèle des transformées opérationnelles qui ont été établies pour la collaboration en temps réel. SOCT2 est le pire algorithme parmi toutes les approches étudiées concernant les besoins en espace mémoire et Logoot est le pire parmi les approches CRDT. De l'autre côté, WOOTO occupe moins de mémoire, il est qualifié comme le plus approprié pour l’édition collaborative en temps-réel sur des applications mobile. À l'avenir, nous prévoyons de faire rejouer cette expérience sur de vrais appareils mobiles afin d'observer les comportements relatifs à l’occupation mémoire, le temps d'exécution CPU et la consommation d’énergie.

6 Références

[1] Cleveland W S. G. Tammaro, J. N. Mosier, N. C. Goodwin, et G. Spitz. *Collaborative Writing Is Hard to Support: A Field Study of Collaborative Writing*. Computer-Supported Cooperative Work, 6(1): 19-51, 1997.

[2] C. A. Ellis et S. J. Gibbs. *Concurrency Control in Groupware Systems*. SIGMOD Record: Proceedings of the ACM SIGMOD Conference on the Management of Data - SIGMOD '89, 18(2):399-407, Mai 1989.

[3] M. Ressel, D. Nitsche-Ruhland, et R. Gunzenhauser. *An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors*. Proceedings of the ACM

Conference on Computer-Supported Cooperative Work - CSCW '96, pp. 288-297, Boston, MA, États-Unis, Novembre 1996. ACM Press.

[4] P. A. Bernstein, V. Hadzilacos, et N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Boston, MA, États-Unis, 1987.

[5] S. Buchegger, D. Schioberg, L. H. Vu, et A. Datta. *PeerSoN: P2P Social Networking - Experiences and Insights*. Proceedings of the Second ACM EuroSys Workshop on Social Network Systems – SNS 2009, pp. 46-52, Nuremberg, Allemagne, Mars 2009. ACM Press.

[6] B. Collins-Sussman, B. W. Fitzpatrick, et C. M. Pilato. *Version control with Subversion*. O'Reilly & Associates, Inc., 2004.

[7] C. Sun et C. Ellis. *Operational Transformation in Real-Time Group Editors: Issues, Algorithms and Achievements*. Proceedings of the ACM Conference on Computer-Supported Cooperative Work – CSCW '98, pp. 59-68, Seattle, WA, États-Unis, Novembre 1998. ACM Press.

[8] D. A. Nichols, P. Curtis, M. Dixon, et J. Lamping. *High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System*. Proceedings of the 8th Annual ACM Symposium on User interface and Software Technology - UIST '95, pp. 111-120, Pittsburgh, PA, États-Unis, Novembre 1995. ACM Press.

[9] M. Suleiman, M. Cart, et J. Ferrié. *Serialization of Concurrent Operations in a Distributed Collaborative Environment*. Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP '97, pp. 435-445, Phoenix, AZ, États-Unis, Novembre 1997. ACM Press.

[10] N. Vidot, M. Cart, J. Ferrié, et M. Suleiman. *Copies Convergence in a Distributed Real-Time Collaborative Environment*. Proceedings of the ACM Conference on Computer-Supported Cooperative Work – CSCW 2000, pp. 171-180, Philadelphia, PA, États-Unis, Décembre 2000. ACM Press.

[11] C. Sun, X. Jia, Y. Zhang, Y. Yang, et D. Chen. *Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems*. ACM Transactions on Computer-Human Interaction, 5(1):63-108, Mars 1998.

[12] G. Oster, P. Urso, P. Molli, et A. Imine. *Data Consistency for P2P Collaborative Editing*. Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006, pp. 259-267, Banff, AB, Canada, Novembe 2006. ACM Press**.**

[13] N. Preguiça, J. M. Marques, M. Shapiro, et M. Letia. *A Commutative Replicated Data Type for Cooperative Editing*. Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009, pp. 395-403, Montreal, QC, Canada, Juin 2009. IEEE**.**

[14] H.-G. Roh, M. Jeon, J. Kim, et J. Lee. *Replicated Abstract Data Types: Building Blocks for Collaborative Applications*. Journal of Parallel and Distributed Computing, 71(3):354-368, 2011**.**

[15] S. Weiss, P. Urso, et P. Molli. *Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks*. Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009, pp. 404-412, Montreal, QC, Canada, Juin 2009. IEEE Computer Society**.**

[16] S. Weiss, P. Urso, et P. Molli. *Wooki: A P2P Wiki-based Collaborative Writing Tool*. Proceedings of the International Conference on Web Information Systems Engineering - WISE 2007, pp. 503-512, Nancy, France, Décembre 2007. Springer-Verlag**.**

[17] A. Imine, P. Molli, G. Oster, et M. Rusinowitch. *Proving Correctness of Transformation Functions in Real-Time Groupware*. Proceedings of the European Conference on Computer-Supported Cooperative Work - ECSCW 2003, pp. 277-293, Helsinki, Finland, Septembre 2003. Kluwer Academic Publishers.

[18] G. Oster, P. Molli, P. Urso, et A. Imine. *Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems*. Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006, pp. 1-10, Atlanta, GA, États-Unis, Novembre 2006. IEEE.

[19] M. Ahmed-Nacer, G. Oster, P. Urso, C.-L. Ignat et H.-G. Roh. *Evaluating CRDTs for Real-time Document Editing*. ACM Symposium on Document Engineering, Septembre 2011, San Francisco, CA, États-Unis.

[20] Shao. Bin, Li. Du et Gu. Ning. *A Fast Operational Transformation Algorithm for Mobile and Asynchronous Collaboration*. IEEE Transactions on Parallel and Distributed Systems - 2010.

[21] D. Li et R. Li. *A Performance Study of Group Editing Algorithms*. Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS 2006, pp. 300-307, Minneapolis, MN, États-Unis, Juillet 2006. IEEE.