Documents collaboratifs mobiles et nuage

Ahmed-Nacer Mehdi[*]
Université de Lorraine
LORIA, INRIA

Pascal Urso[†]
Université de Lorraine
LORIA, INRIA

Nuno Preguiça[‡]
Universidade Nova de Lisboa
CITI, FCT

Valter Balegas[§]
Universidade Nova de Lisboa
CITI, FCT

Résumé

Aujourd'hui, un grand nombre d'applications d'édition de documents ont été développés, notamment pour les appareils mobiles. Certains d'entre eux sont déployés dans les nuages tel que Google Drive et SkyDrive de Microsoft.

Dans le cadre d'édition collaborative, chaque utilisateur possède 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. Cependant, la performance des dispositifs mobiles sont limitées.

Un grand nombre de système collaborative utilise les Transformées Opérationnelles (OT) pour contrôler l'édition concurrente. Ce type d'approches ne s'adaptent pas bien dans les environnements pair-à-pair où elle ne passe pas à l'échelle. Récemment, une nouvelle approche dite Commutative Replicated Data Types (CRDT) a été proposée comme substitut de l'approche OT pour assurer la cohérence des données dans les réseaux pair-à-pair.

Dans cet article, nous proposons une architecture pour tirer profit des deux approches - OT et CRDT - et pour améliorer les performances des applications d'édition collaborative mobile.

Categories and Subject Descriptors

I.7.1 [**Document and Text Processing**]: Document and Text Editing; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

[*]mahmedna@loria.fr
[†]pascal.urso@loria.fr
[‡]nuno.preguica@fct.unl.pt
[§]v.sousa@campus.fct.unl.pt

Ubimob'14 5-6 juin 2014, Sophia Antipolis (France)

Mots-clés

Édition collaborative, Types de données répliqués, Transformées Opérationnelles, Cloud, CRDT

1. INTRODUCTION

Les dispositifs mobiles tels que les ordinateurs portables, les tablettes, netbook et les téléphones intelligents 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 de documents 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 GoogleDrive 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 mobile doit soigneusement prendre en compte les contraintes de ces systèmes : connexion intermittente, mémoire limitée, puissance de traitement limitée, bande passante limitée, autonomie de la batterie, etc.

Aujourd'hui, un grand nombre d'applications permettent l'édition de documents sur dispositifs mobiles. Ces applications sont dédiées au traitement de textes, tableurs, prise de notes, dessins, etc. – et certaines, tel Google Drive ou SkyDrive de Microsoft, permettent la collaboration entre plusieurs utilisateurs. La plupart de ces applications proposent à l'utilisateur d'héberger leurs documents dans une infrastructure dite "dans le nuage". Pour assurer l'accès imédiat aux documents et parfois une édition en mode déconnecté, chaque dispositif mobile hébèrge une réplique locale des documents. Quand un utilisateur génère une action, cette dernière est transformée en un ensemble d'opérations qui est exécutée localement et envoyée aux autres copies.

L'application est alors chargée de maintenir la cohérence entre les différentes répliques. Même pour les applications ne proposant pas d'édition collaborative, ces répliques sont nombreuses :

- une réplique par dispositif mobile (et fixe) d'un même utilisateur,
- plusieurs utilisateurs dans le cadre d'une application collaborative,
- plusieurs répliques par centre de données (cloud data center) afin d'assurer la répartion de charge et la tolérance aux pannes

– plusieurs centres de donnés pour supporter les partitions réseaux et les pannes massives.

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. Un tel mécanisme est nécessairement optimiste afin de permettre le travail déconnecté et la tolérance aux pannes d'après le théorème CAP [3, 6]. Deux principaux types d'approches existent pour permettre la réplication optimiste de documents.

L'approche dite transformées opérationnelles (OT) a été proposée comme un mécanisme approprié pour la collaboration en temps réel [5, 9, 11]. Elle assure la cohérence des données en adaptant les paramètres des opérations distantes afin de prendre en compte les effets des opérations concomitantes. Il existe différentes architectures de déploiement de systèmes OT. Les architectures décentralisées sont complexes à mettre en oeuvre et nécessite de coûteux méchanismes – tels des vecteurs d'horloges et un historique d'opérations non purgeable – qui ne passent pas à l'échelle surtout sur dispositif mobile [1]. Les architectures centralisées – tel l'algorithme Jupiter [7] utilisé dans Google Drive – tolère un nombre massif de clients mais requièrent un séquenceur central [12]. A l'aide d'un consensus distribué[4], un séquenceur central peut être mis en place dans le cadre d'un centre de données, mais aucune solution multi-centre de données ne tolère les partitions.

Une nouvelle approche, dite Commutative Replicated Data Types (CRDT)[8, 10], 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 opérationnelles, les CRDTs assurent la convergence des documents sans bloquer les opérations du client et sans consensus. De plus, elle n'utilise pas l'historique des opérations pour détecter la concurrence entre les opérations afin d'assurer la cohérence. Cependant, ces approches utilisent des méta-données de taille non négligeable pour un dispositif mobile, i.e. un identifiant de taille non-bornée par caractère. De plus les performances de ces approches restent inférieures aux approches OT centralisées.

Dans cet article nous proposons une architecture en combinant les deux protocoles – OT et CRDT –, de telle façon à éviter le consensus inter centre de données, tirer profit des deux approches et satisfaire l'utilisateur tout en respectant les contraintes des dispositifs mobiles.

2. ARCHITECTURE

L'architecture de notre système considère plusieurs centres de données. Nous différencions les noeuds serveur qui sont hébergés dans un centre de données et les noeuds clients qui sont les dispositifs (mobile ou fixe) de l'utilisateur qui sont répartis sur l'ensemble de la planète. Afin de permettre la répartition de charge et de tolérer les pannes, plusieurs serveurs identiques sont déployées dans un centre de données. La cohérence entre les serveurs dans un centre de données particulier peut être assurée par un algorithme de type consensus [4]. Dans la suite de ce document, nous considérons l'ensemble des serveurs d'un centre de données comme un seul noeud serveur.

L'idée de base de notre architecture est

– d'utiliser un algorithme centralisé OT entre les clients et le centre de donnée de leur choix,
– de coupler fortement chaque serveur central OT à une réplique CRDT
– d'assurer la cohérence inter centre de données de manière pair-à-pair à l'aide du mécanisme CRDT

La figure 1 montre l'architecture de notre système : Les clients exécutent les opérations localement. Le séquenceur OT se trouve dans le même centre de données que le serveur et gère l'ordre entre les opérations générées par le client. Une fois que le serveur reçoit et intégre l'opération OT, il génère une nouvelle opération CRDT correspondante et dissémine cette opération aux autres serveurs. Le serveur qui reçoit l'opération CRDT, génère une nouvelle opération OT correspondante et il l'envoie à ses clients. Les deux opérations sont exécutées en isolation pour assurer que les deux états OT et CRDT dans le serveur soient équivalents.

Ainsi, les clients qui utilisent un dispositif mobile ou fixe exécutent un client OT. Les serveurs qui sont dans le centre de données, exécutent deux protocoles : l'algorithme OT est utilisé pour accepter les opérations de ses clients et qui sont dans la même région ; l'algorithme CRDT est utilisé pour envoyer et recevoir les opérations entre les centres de données.

L'avantage de cette approche est que le dispositif mobile du client reste léger et le flux de données sur le réseau est réduit, car les clients utilisent algorithme OT centralisé.

Les opérations sont propagées entre les data-center en utilisant l'approche CRDT, ce qui évite l'utilisation d'un consensus entre les serveurs. L'inconvénient est que les serveurs doivent exécuter chaque opération deux fois, mais nous supposons que les serveurs dans les data-center possède une puissance de calcul raisonnable.

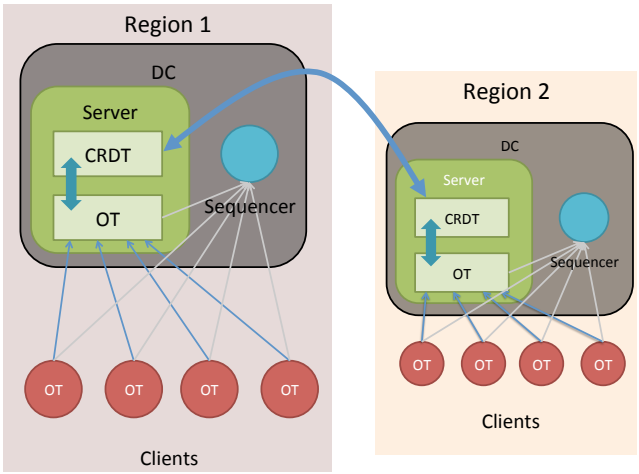
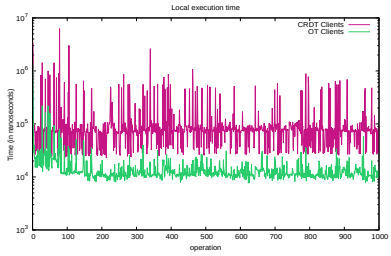


Figure 1: Architecture OT/CRDT.

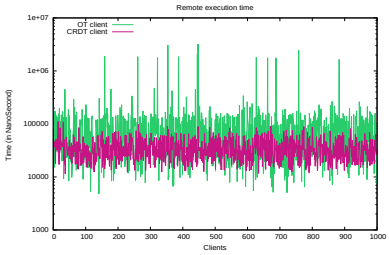
Travail déconnecté

Dans les dispositifs mobiles, il est important de pouvoir travailler de manière déconnectée. Heureusement, certains algorithmes OT, comme SOCT4, [1] permettent d'exécuter des opérations locales sans accès au séquenceur. En effet les opération produites hors-ligne sont mise de côté le temps de recevoir leur estampille. La seule contrainte de tel algorithmes est d'empêcher l'intégration d'opération distantes tant que des opérations locales n'ont pas reçu d'estampille.

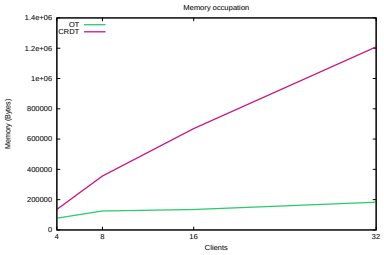
1. Contrairement à l'algorithme Jupiter utilisé dans Google Drive



(a) Temps d'exécution local



(b) Temps d'intégration



(c) L'occupation mémoire

Ceci n'est pas génant pour le travail déconnecté, mais empêche de travailler dans un sous groupe isolé du réseaux, ce que seul les algorithmes réellement pair-à-pair autorisent. Un système utilisant SOCT4 pour le travail connecté et déconnecté à été décrit dans [2].

3. ÉVALUATION EXPÉRIMENTALE

Pour évaluer les performances des algorithmes, nous avons implémenté l'architecture décrite précédemment en Java, et nous avons intégré un simulateur pour générer des opérations. Nous avons simulé une collaboration entre 64 clients suivant des caractéristiques obtenues dans [1]. Nous avons déployé cette architecture sur des machines réel sur Amazon. Nous avons créé deux serveurs et 64 clients dispersés dans différentes régions. Les serveurs en Irlande et Californie et utilise des machines moyenne. Alors que les clients sont dispersés dans 4 régions : Tokyo, Virginia, Singapore et Sydney et utilise des micro-machines pour simuler les clients. Le framework utilise java.lang.System.nanoTime() pour mesurer le temps d'exécution et et l'interface serialization de Java pour estimer la mémoire occupée dans chaque périphérique.

Les algorithmes que nous avons utilisé dans cette expérimentation sont l'algorithme OT SOCT4 et l'algorithme CRDT Logoot.

SOCT4 [12] est un algorithme de type transformées opérationnelles. Lorsqu'un utilisateur génère une opération, celle-ci est immédiatement exécutée, puis le client demande une estampille pour cette opération. Une fois l'estampille reçue, l'opération est placée dans une file d'attente. Elle ne sera envoyée au serveur – et donc aux autres clients – qu'une fois prête, c'est à dire lorsque son estampille sera immédiatement supérieure à l'estampille de la dernière opération envoyée ou reçue. Ce mécanisme permet de délivrer sur chaque site les opérations suivant l'ordre donné par le séquenceur.

Quand une opération distante est reçue, SOCT4 transforme cette opération avec les opérations locales concurrentes, i.e. les opérations locales en attente. Pour que l'algorithme SOCT4 assure la convergence, il faut que les transformées opérationnelles respectent la condition C1 [12].

Logoot [13] est une approche CRDT pour l'édition de documents texte. 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 SOCT4, Logoot sauvegarde dans son modèle tous les identifiants. Cela représente un inconvénient majeur pour les appareils mobile.

L'algorithme SOCT4 est plus performant que l'algorithme Logoot en temps d'exécution local (figure 2a). En effet, l'algorithme Logoot a besoin de générer un identifiant unique pour chaque opération. Alors que, SOCT4 exécute directement l'opération et ne nécessite aucun identifiant.

Lors de l'intégration d'une opération (figure 2b), l'algorithme SOCT4 est moins performant que Logoot, mais cela reste acceptable et ne dépasse pas 5 ms. En effet, l'algorithme SOCT4 a besoin de parcourir un log pour détecter les opérations concurrente et les transformer pour les appliquer au bon endroit. Cependant, l'algorithme Logoot utilise la recherche dichotomique pour trouver la position correcte.

La mémoire occupée(figure 2c) par Logoot connaît une croissance rapide qui peut être problématique, particulièrement sur dispositifs mobiles. En effet, Logoot stocke pour chaque caractère un identifiant. De plus, la taille de l'identifiant Logoot est relatif aux nombres d'opérations concurrente et grossit au fur à mesure du temps. Pour cette raison, plus le nombre de clients devient important et plus la mémoire occupée est importante. L'algorithme SOCT4 sauvegarde dans son journal uniquement ses propres opérations, le temps qu'elle soient émises vers le serveur SCOT4. Une fois émise par le client, il peut les enlever de son journal car il ne recevra plus aucune opération précédente.

4. CONCLUSION

Dans ce papier, nous avons proposé une architecture pour les applications d'édition collaborative allégeant fortement la charge sur les dispositifs clients et donc particulièrement destinées aux dispositifs mobiles.

Nous avons constaté que, les applications d'édition collaborative peuvent être améliorées par la fusion des deux approches OT et CRDT. Sur cette base, les applications respectent les contraintes des appareils mobiles tel que la mémoire occupée et la consommation de CPU, supportent plus de clients, permettent de réduire le flux dans le réseau et le temps de latence entre les clients.

De plus, l'architecture proposée ne nécessite pas de consensus entre les centre de données pour synchroniser le séquenceur distribué comme dans les approches grand public existantes.

5. REFERENCES

[1] M. Ahmed-Nacer, C.-L. Ignat, G. Oster, H.-G. Roh, and P. Urso. Evaluating crdts for real-time document editing. In ACM, editor, *ACM Symposium on Document Engineering*, page 10 pages, San Francisco, CA, USA, september 2011.

[2] A. Bouazza and P. Molli. Unifying coupled and uncoupled collaborative work in virtual teams. In *2000 ACM Conference on Computer Supported Cooperative Work (CSW'2000) Workshop on collaborative editing systems*, page 6 p, Philadelphia, Pennsylvania, USA, December 2000. Colloque avec actes et comité de lecture. internationale. A00-R-224 || bouazza00a A00-R-224 || bouazza00a.

[3] E. A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, pages 7–, New York, NY, USA, 2000. ACM.

[4] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI*, pages 335–350. USENIX Association, 2006.

[5] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In J. Clifford, B. G. Lindsay, and D. Maier, editors, *SIGMOD Conference*, pages 399–407. ACM Press, 1989.

[6] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33 :51–59, June 2002.

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

[8] G. Oster, P. Urso, P. Molli, and A. 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, nov 2006. ACM Press.

[9] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *CSCW*, pages 288–297, 1996.

[10] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In X. Défago, F. Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6976, pages 386–400, Grenoble, France, October 2011.

[11] C. Sun and C. A. Ellis. Operational transformation in real-time group editors : Issues, algorithms, and achievements. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'98*, pages 59–68, New York, New York, États-Unis, November 1998. ACM Press.

[12] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 171–180, New York, NY, USA, 2000. ACM.

[13] S. Weiss, P. Urso, and P. Molli. Logoot : A scalable optimistic replication algorithm for collaborative editing on p2p networks. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009)*, pages 404 –412, Montréal, Québec, Canada, jun. 2009. IEEE Computer Society.