Estampillage et Journalisation P2P pour XWiki

Mounir Tlili, William Kokou Dedzoe, Esther Pacitti, Patrick Valduriez, Reza Akbarinia, Ludovic Dubost, Sergiu Dumitriu, Stéphane Laurière, Gérôme Canals, Pascal Molli, et al.

Estampillage et Journalisation P2P pour XWiki

Mounir Tlili, W. Kokou Dedzoé, Esther Pacitti, Patrick Valduriez
Atlas team, INRIA and LINA, University of Nantes, France
{FirstName.LastName@univ-nantes.fr, Patrick.Valduriez@inria.fr}

Reza Akbarinia
School of Computer Science University of Waterloo, Canada
rakbarin@cs.uwaterloo.ca

Ludovic Dubost, Sergiu Dumitriu, Stéphane Laurière
XWiki, France
Franceprenom@xwiki.com

Gérôme Canals, Pascal Molli, Julien Maire
Ecoo team, LORIA-INRIA
{FirstName.LastName}@loria.fr

Résumé

Les systèmes pair-à-pair sont de plus en plus utilisés pour développer des applications distribuées au sein des entreprises. Les réseaux pair-à-pair permettent de construire des applications fiables, performantes, disponibles et passant à l'échelle en répliquant les données sur plusieurs pairs du réseau.

Dans cet article nous nous intéressons à la réplication des données de l'application XWiki dans un réseau pair-à-pair. Pour ce faire, nous proposons un nouveau mécanisme de réplication optimiste combinant un algorithme de réconciliation de données basé sur une approche de transformées opérationnelles et une extension du service KTS qui intègre un mécanisme d'estampillage fiable et réparti fonctionnant sur un modèle de réseau à base de DHT pour gérer la fraîcheur des répliques.

Le travail présenté est en partie financé par l'Agence Nationale de la Recherche dans le cadre du projet *RNTL XWiki Concerto* (2007-2009), projet de wiki P2P supportant la mobilité.

Thème

H.2 [Database management]: Distributed databases, concurrency

Termes Générale

Algorithms, Performance, Design

Mot-Clés

bases de données, réplication optimiste, système pair-à-pair, réconciliation, DHT.

1. INTRODUCTION

Les systèmes pair-à-pair (P2P) sont en plein essor depuis maintenant plusieurs années. Ce paradigme permet de concevoir des systèmes de très grande taille à forte disponibilité et à faible coût. En effet, les réseaux P2P sont des réseaux overlay, décentralisés, où tous les nœuds, appelés pairs, jouent à la fois le rôle de serveur et de client. L'organisation dans un tel réseau repose sur l'ensemble des pairs, il n'y a pas d'entité chargée d'administrer ce réseau.

Une classe d'application importante dans les systèmes P2P concerne les applications collaboratives où un grand nombre d'utilisateurs doivent pouvoir travailler sur les mêmes données (documents, fichiers, etc.) en parallèle depuis leurs postes de travail. Un exemple de telle application est XWiki [1] [2], un wiki de seconde génération capable de gérer des données structurées et des données non structurées. Un wiki est un outil d'édition de documents en ligne dans lequel les utilisateurs peuvent facilement tisser des liens entre les documents. Il permet la création et la modification de documents à travers une interface Web. Lors de la navigation dans le contenu d'un wiki, un utilisateur peut à tout moment entrer en mode édition pour modifier le contenu de la page. Lorsqu'il a terminé, il peut sauvegarder le nouveau contenu qui vient alors remplacer l'ancienne valeur de la page.

L'architecture actuelle de XWiki est basée sur une architecture classique client-serveur : un serveur détient les données et les utilisateurs consultent et éditent les pages à travers un navigateur Web qui interagit avec le serveur. Cette architecture, malgré sa simplicité et sa popularité, a des limites : par exemple, l'utilisation d'un serveur central est un point de faiblesse qui rend le système vulnérable aux pannes. De plus, le système impose d'être connecté au serveur pour réaliser des opérations d'édition.

L'objective du projet *ANR XWiki Concerto* est de concevoir une architecture XWiki P2P qui dépasse ces limites afin de passer à l'échelle en nombre d'utilisateurs et de supporter la mobilité des clients. Une telle application demande des capacités générales de réplication avec différents niveaux de granularité (ligne, mot) et un mode multi-maître où plusieurs répliques d'une même donnée peuvent être mises à jour par plusieurs pairs en parallèle.

La réplication multi-maître offre de bons avantages de performance et de disponibilité. Cependant, les mises à jour de la même donnée par différents pairs peuvent créer des divergences des copies. La solution de réplication optimiste est alors appropriée aux systèmes P2P [3] car elle permet aux répliques d'être mises à jour indépendamment et de rester divergentes jusqu'à une étape de réconciliation. Cependant, elle doit être étendue pour aborder l'autonomie des pairs et la très grande échelle du réseau P2P.

Le défi est donc de construire un mécanisme de réplication multi-maître supportant la modification simultanée de différentes copies d'un même objet avec fusion de données adaptée aux besoins de l'application XWiki. Un tel mécanisme permet d'améliorer la disponibilité de données. Nous proposons dans ce papier une solution nouvelle à ce problème. Elle combine un algorithme de réconciliation de données basé sur une approche de transformées

opérationnelles (OT) [4] et une extension du service KTS [5], que nous avons appelé KTS-Log, pour gérer la fraîcheur des répliques dans une table de hachage distribuée (DHT). L'algorithme de réconciliation que nous avons utilisé nécessite un service de gestion d'estampilles continues et fiable, nécessaires pour ordonnancer les actions (les modifications sur les documents XWiki) stockées dans les journaux. KTS-Log, grâce à son utilisation d'une DHT, permet de gérer des estampilles de façon complètement distribuée.

Le plan de notre article est le suivant. Dans une première partie, nous décrivons l'architecture de notre solution à savoir l'algorithme de réplication utilisé. Dans une deuxième partie nous présentons l'implémentation de notre système. Enfin, dans une troisième partie nous présentons quelques scenarios que nous avons utilisés pour valider notre prototype.

2. ARCHITECTURE ET SPECIFICATION

Cette partie est consacrée à la description de l'architecture conçue pour la réplication des données sur un réseau P2P, et à la spécification des différents composants de cette architecture.

2.1 Architecture Générale

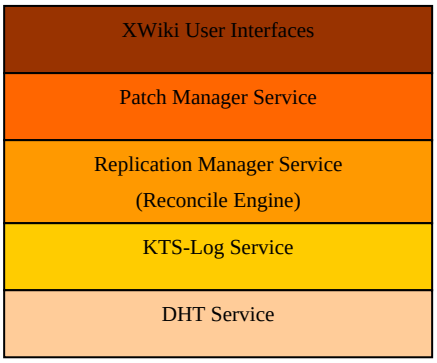


Figure 1. Architecture générale du système

La Figure 1 illustre l'architecture de notre système. Il s'agit d'une architecture multi-couche à base de services. Pour associer l'application XWiki au service de réplication nous avons utilisé une API de type Web service afin d'éviter la modification du code de XWiki. Dans cette architecture, XWiki est vu comme un service Web capable de produire des modifications (Patch[1]) sur un ensemble de pages wiki, et d'appliquer des modifications sur un ensemble de pages wiki. Le service de réplication est donc en charge de répliquer une modification produite par un service Web XWiki en l'appliquant (après éventuellement transformation) à un ensemble de services Web XWiki distants. Une modification correspond aux changements produits par un utilisateur lorsqu'il sauvegarde une page.

Nous distinguons donc quatre couches de services dans cette architecture : Patch Manager Service, Replication Manager Service, KTS-Log Service, et DHT Service.

[1] Un patch contient une séquence d'opérations ordonnée chronologiquement.

Patch Manager Service: Ce service assure la circulation de patchs échangés entre l'application cliente XWiki et le composant de réplication. Ce service génère, applique, et fournit des patchs.

Replication Manager Service: Ce service a pour fonction d'augmenter la disponibilité des données en stockant plusieurs copies de la même donnée sur plusieurs nœuds. Le Replication Manager Service n'échange pas directement les opérations (Patch) avec les autres applications clientes, il doit d'abord les faire transiter par le service KTS-Log. Ce dernier estampille les opérations selon un ordre total avant de les stocker dans plusieurs nœuds dans la DHT. S'il y a des nouvelles opérations dans la DHT, Le Replication Manager Service doit d'abord recevoir toutes les opérations estampillées et les envoyer ensuite à son moteur de réconciliation. Il est ensuite possible d'estampiller les opérations locales et de les stocker dans la DHT.

KTS-Log Service: Ce service propose un mécanisme d'estampillage fiable, réparti et fonctionnant sur un réseau P2P utilisant une DHT. Il sert non seulement à estampiller les opérations reçues mais aussi à les faire stocker dans la DHT.

DHT Service: Le DHT Service fournit une solution intéressante pour le passage à l'échelle dans le stockage des opérations *(Log)* dans un système P2P. Une table de hachage est une structure de données qui associe une clé à une ressource. Chaque clé de la table est le résultat d'une fonction de hachage appliquée à un élément de la ressource, comme par exemple, son nom. La fonction de hachage garantit que pour deux ressources différentes, les clés générées le seront aussi. Ainsi, l'unicité de la clé permet d'identifier et de retrouver de manière fiable la ressource à laquelle elle est associée. Les opérations de base réalisables sur les DHTs sont les opérations *lookup(clé)* et *put(clé,valeur)*, permettant respectivement de récupérer la valeur associée à la clé et de stocker une clé et sa valeur associée dans la table de hachage distribuée.

2.2 Notion de Post Groupe

Dans notre architecture, nous attribuons pour chaque document XWiki un ensemble de nœuds noté Post Groupe: Master(key), Master-Succ(key), Log-Peers(key-ts[2]). (voir Figure 2)

Master Peers : ce sont les nœuds responsables de la génération des estampilles. Ces nœuds doivent disposer au préalable d'un ensemble de fonctions de hachages utilisées pour la réplication des patchs dans la DHT.

De façon distribuée, les Master Peers permettent la gestion des estampilles de différentes clés et ensuite la réplication des patchs estampillés dans les **Log Peers**.

Les opérations principales de KTS-Log sont ***gen_ts(key)*** qui produit des estampilles monotonement croissantes pour les clé et ***last_ts(key)*** qui retourne la dernière estampille générée pour la clé Key.

Ayant la dernière valeur d'estampille ***ts***, l'utilisateur peut rechercher facilement sur la DHT les patchs manquants en utilisant la fonction ***: Get(key+ts).***

[2] Clé de document + estampille

Master-Succ: remplace le Master en cas de panne et tient la responsabilité des répliques pour la dernière valeur d'estampille généré ***last-ts***.

Log-Peers: ce sont les nœuds responsables pour le stockage des patches estampillés de différents documents XWiki.

Une opération est signalée répliquée dans les Log Peers par l'exécution des opérations suivantes: Put(h_1(key-ts),Patch), Put(h_2(key-ts),Patch)…Put(h_n(key-ts),Patch).

Log-Peers-Succ: remplace les Log-Peers en cas de panne.

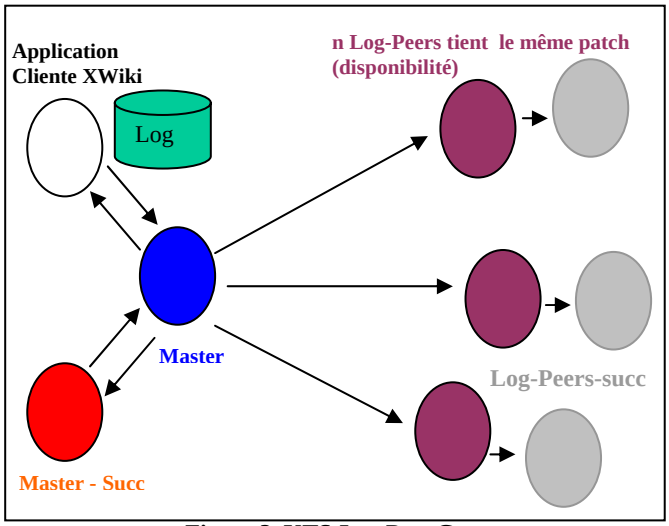


Figure 2. KTS-Log Post Groupe

2.3 Hypothèses

La gestion de la réplication dans un réseau P2P supportant la modification simultanée de différentes copies d'un même objet est un défi, étant donné le comportement dynamique des pairs. Afin de garantir la cohérence et l'ordre total des estampilles nos algorithmes respectent certaines hypothèses acceptables et cohérentes avec KTS [5].

- Chaque nœud a un identifiant unique PeerID.
- Chaque nœud tient un ensemble de fonctions de hachages utilisées pour répliquer les données XWiki dans la DHT et pour la récupération des estampilles.
- Chaque document XWiki a un unique identifiant DocID.
- Il ne peut pas y avoir plus de n pannes simultanées des Log -Peers de mêmes clés.
- Deux nœuds successeurs ne peuvent pas tomber en panne en même temps.
- Durant la stabilisation des Masters Peers, aucune estampille n'est délivrée.
- Si un nœud du réseau tombe en panne, son successeur le remplace automatiquement.

2.4 Algorithme de réplication

Nous décrivons dans cette partie le principe de fonctionnement de notre algorithme de réplication proposé pour garantir la disponibilité de données.

Principe de fonctionnement : Dans notre architecture, chaque site disposant d'une copie des données partagées génère des opérations et les intègre localement avant de les publier dans la DHT sous forme de patch. Pour envoyer une opération, il faut d'abord recevoir toutes les opérations estampillées et les intégrer localement. Il est ensuite possible d'estampiller les opérations locales et de les envoyer.

Le principe de fonctionnement de l'algorithme est basé sur les Master Peers qui délivrent les estampilles à chaque opération. Les opérations sont donc estampillées selon un ordre total.

Supposons qu'une application cliente s'exécute sur chaque peer. Les applications clientes ne s'échangent pas directement les opérations, elles doivent les faire transiter par Les estampilleurs « Master Peers » qui à leur tour les font stocker dans les Log-Peers en utilisant un ensemble de fonction de hachage. Ces Log-Peers maintiennent, de manière persistante, les opérations publiées par les différents sites dans l'ordre des estampilles.

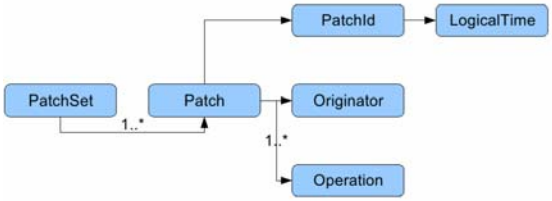
3. PROTOTYPE

L'implémentation actuelle de notre prototype est basée sur Open Chord [6] comme protocole de DHT. Open Chord est une implémentation en Java de Chord [7], proposée par des chercheurs de l'Université de Bamberg en Allemagne.

Dans notre système, les pairs de DHT sont implémentés comme des objets Java. Chaque objet contient le code dont il a besoin afin d'implémenter le service KTS-Log. La communication entre les objets KTS-Log est assurée via le protocole Java RMI [8], permettant donc l'invocation des méthodes sur des objets distants.

Dans notre prototype (voir Figure 3), nous utilisons les interfaces Web de l'application XWiki permettant donc aux utilisateurs du système de manipuler les contenues des pages XWiki (modifier, ajouter ou supprimer des données).

Le service de gestion de patchs met en œuvre le modèle de



données de la Figure 4. Il génère des fichiers XML décrivant les opérations atomiques effectuées sur chaque serveur XWiki et les met à disposition du service par l'intermédiaire de l'API REST [9]. Les patchs sont accompagnés d'une signature garantissant l'intégrité des données lors du transfert.

Figure 4.Modèle de données du service de gestion de patchs

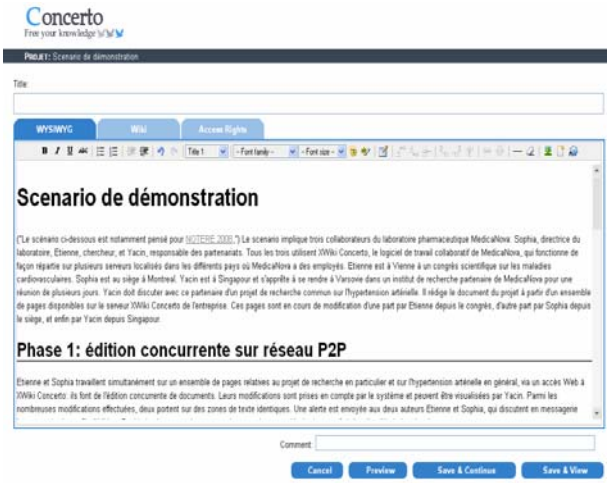


Figure 3. Exemple d'édition d'un document XWiki

4. DEMONSTRATION

Nous présentons le fonctionnement de prototype à travers une démonstration mettant en œuvre le scenario suivant.

Le scenario implique trois collaborateurs du laboratoire pharmaceutique MedicaNova: Sophia, directrice du laboratoire, Etienne, chercheur, et Yacin, responsable des partenariats. Tous les trois utilisent XWiki Concerto, le logiciel de travail collaboratif de MedicaNova, qui fonctionne de façon répartie sur plusieurs serveurs localisés dans les différents pays où MedicaNova a des employés. Etienne est à Vienne à un congrès scientifique sur les maladies cardiovasculaires. Sophia est au siège à Montreal. Yacin est à Singapour et s'apprête à se rendre à Varsovie dans un institut de recherche partenaire de MedicaNova pour une réunion de plusieurs jours. Yacin doit discuter avec ce partenaire d'un projet de recherche commun sur l'hypertension artérielle. Il rédige le document du projet à partir d'un ensemble de pages disponibles sur le serveur XWiki Concerto de l'entreprise. Ces pages sont en cours de modification d'une part par Etienne depuis le congrès, d'autre part par Sophia depuis le siège, et enfin par Yacin depuis Singapour.

4.1 Edition concurrente sur réseau P2P

Etienne et Sophia travaillent simultanément sur un ensemble de pages relatives au projet de recherche en particulier et sur l'hypertension artérielle en général, via un accès Web à XWiki Concerto: ils font de l'édition concurrente de documents. Leurs modifications sont prises en compte par le système et peuvent être visualisées par Yacin. Parmi les nombreuses modifications effectuées, deux portent sur des zones de texte identiques. Une alerte est envoyée aux deux auteurs Etienne et Sophia, qui discutent en messagerie instantanée du conflit d'édition. Sophia le résout ensuite en connaissance de cause. Yacin de son côté visualise l'évolution des documents.

4.2 Passage en mode déconnecté d'un des nœuds du réseau

Yacin s'apprête à quitter Singapour pour rejoindre Hambourg par avion. Il ne disposera pas de connexion à Internet de bonne qualité entre les deux villes. Il passe donc en mode XWiki Concerto déconnecté, pour pouvoir continuer à éditer les documents localement et synchroniser ultérieurement ses changements avec ceux qui auront été apportés entre temps par Sophia et Etienne.

4.3 Réconciliation des données

Une fois qu'il dispose de nouveau d'une connexion Internet à Varsovie, Yacin se reconnecte au réseau MedicaNova et lance une opération de synchronisation. Au moment de la reconnexion, comme Yacin a changé de réseau, un ensemble de vérifications est effectué avant que Yacin obtienne l'autorisation de transmettre ses changements locaux et de télécharger les modifications distantes. L'ensemble de ses modifications est intégré au serveur, et propagé vers les serveurs auxquels sont connectés Etienne et Sophia. Pour chacune des pages modifiées, les trois collaborateurs peuvent visualiser l'historique des modifications. Yacin prend connaissance de toutes les dernières modifications apportées par ses collègues, et peut donc produire un document à jour.

5. CONCLUSION

Dans ce document, nous avons présenté notre solution de réplication optimiste chargée de faire fonctionner l'application XWiki sur un réseau P2P à base de DHT, et un scenario de démonstration mettant en pratique le système complet.

Pour ordonner les opérations publiées par les utilisateurs de système, nous avons utilisé un mécanisme d'estampillage fiable qui permet de garantir l'ordre total des estampilles.

6. REFERENCES

[1] XWiki: http://www.xwiki.org.

[2] XWiki Concerto: http://concerto.xwiki.com.

[3] E. Pacitti, O. Dedieu. Algorithms for optimistic replication on the Web. *Journal of the Brazilian Computing Society*, 8(2): 179-183, 2002.

[4] P. Molli, G. Oster, H. Skaf-Molli, A. Imine. Using the transformational approach to build a safe and generic data synchronizer. *ACM SIGGROUP Conference on Supporting Group Work, (GROUP)*, 212-220, 2003.

[5] R. Akbarinia, E. Pacitti, P. Valduriez. Data Currency in Replicated DHTs. *ACM SIGMOD Int. Conf. on Management of Data*, Beijing, China, 211-222, 2007.

[6] Open Chord version 1.0.2 User's Manual. http://www.uni-bamberg.de

[7] I. Stoica, R. Morris, D.R. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In Proc. of the ACM SIGCOMM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications, pages 149-160, San Diego, California, August 2001.

[8] Java RMI. http://java.sun.com.

[9] http://dev.xwiki.org/xwiki/bin/view/Design/XWikiPatchService.