Vers une cohérence causale évolutive sans chaînes de ralentissements

Ilyas Toumlilt, Alejandro Tomsic, Marc Shapiro

Compas'2017 : Parallélisme/ Architecture / Système
Sophia Antipolis, France, du 27 au 30 juin 2017

Vers une cohérence causale évolutive sans chaînes de ralentissements

Ilyas Toumlilt, Alejandro Z. Tomsic, Marc Shapiro

Sorbonne Universités, UPMC Paris 06, F-75005, Paris, France
CNRS, UMR_7606, LIP6, F-75005, Paris, France
INRIA, Équipe Regal, F-75005, Paris, France
Email : ilyas.toumlilt@lip6.fr

Résumé
Pour augmenter les performances et la disponibilité, les bases de données sont généralement répliquées sur différents serveurs au sein de centres de données à travers le monde. Cependant, la réplication implique quelques problèmes de cohérence. La cohérence causale est un choix intéressant pour construire ce modèle de bases de données. Elle permet aux clients d'observer un état cohérent par rapport à leurs écritures, en incluant les opérations qu'ils ont précédemment observé, tout en minimisant les anomalies pour le programmeur. Cependant, les mises en œuvre actuelles de la cohérence causale sont susceptibles de connaitre des *ralentissements en chaîne* à l'échelle de plusieurs centres de données. Ce papier décrit la conception d'un système causalement cohérent, évolutif, et non exposé à l'impact que peut avoir un serveur lent ou défaillant, sur la disponibilité des données.

Mots-clés : Bases de données distribuées, géo-replication, cohérence causale, disponibilité sous partition.

---

1. Introduction

Les bases de données distribuées sont l'un des piliers fondamentaux des services en ligne à large échelle aujourd'hui. Pour atteindre les performances et la disponibilité, les bases de données sont généralement répliquées sur différents serveurs et sur des centres de données (DC) à travers le monde. Cependant, nous ne pouvons pas parler de réplication sans parler des problèmes de cohérence. Le choix du modèle de cohérence dans un système géo-répliqué est donc critique.

Le théorème CAP [1, 10] montre qu'un système de stockage distribué ne peut assurer à la fois la cohérence et la disponibilité des données, en présence de coupures réseau. Cela a conduit à une scission des bases de données en deux classes : d'un côté celles fournissant une cohérence forte grâce aux mises à jour synchrones, telles que MongoDB [15], ce qui implique une latence élevée, et dont le coût augmente avec le nombre de DC et la portée géographique. Cette classe est appelée "Cohérente sous Partition" (CP, de l'anglais *Consistent under Partition*). de l'autre côté, les bases de données sacrifiant la cohérence pour offrir la disponibilité et un accès parallèle aux ressources [1], permettant ainsi des performances significatives. Cette classe est appelée "Disponible sous Partition" (AP, de l'anglais *Available under Partition*). Toutefois dans AP, les mises à jour simultanées peuvent être observées dans des ordres différents selon les répliques, et ainsi

provoquer des anomalies qui posent problème aux développeurs d'applications (ex. Cassandra [11] ou Dynamo [7]).
La cohérence causale [2] est un modèle intéressant pour construire des bases de données géo-répliquées. La relation de causalité comprend à la fois l'ordre des événements au sein d'un même client et les relations lectures depuis un autre client [12]. Ainsi, une base de données causalement cohérente garantit qu'une écriture ne devient visible que lorsque toutes celles dont elle dépend causalement sont visibles, tout en évitant les longues latences associées aux synchronisations de la cohérence forte.

Le problème exposé dans ce papier et celui des "ralentissements en chaîne". En effet, les systèmes causaux actuels empêchent souvent une réplique d'appliquer une écriture *W*, jusqu'à ce que toutes les écritures précédent *W* causalement avaient été appliquées. Ainsi, la stabilisation du système, ou en d'autres termes, la propagation des mises à jour entre centres de données, génère une latence qui peut accroître selon l'activité du système. Par conséquent, un fragment lent ou défaillant, peut avoir un impact négatif sur l'ensemble du système, ce qui provoque une augmentation de la latence au niveau de la visibilité des mises à jour.

2. Définition : Causalité

La causalité est la relation de cause à effet qui lie deux événements [2]. On dénote l'ordre causale par "<". Pour deux transactions X et Y, si X < Y, on dit que Y dépend de X ou que X est une dépendance de Y.
X < Y si et seulement si l'une des trois règles suivantes est valide :

- File d'exécution : X et Y sont deux opérations dans un seul thread d'exécution, et X se produit avant Y.
- Lecture : X est une opération d'écriture, Y est une opération de lecture, et Y lit la valeur écrite par X.
- Transitivité : Il existe une opération Z telle que X < Z et Z < Y.

Une version A d'un objet a est causalement dépendante d'une version B d'un objet b si l'écriture de A dépend causalement de l'écriture de B.
Un centre de données est causalement cohérent si, lorsqu'une certaine version d'un objet est visible à un client, alors toutes ses dépendances causales sont aussi visibles.

3. Motivation

Dans un système géo-répliqué, une défaillance sur un serveur affectera inévitablement les performances de ce serveur. Une cascade de ralentissement se produit lorsque la défaillance se répand pour affecter d'autres partitions [6].

Les systèmes causalement cohérents récents, sont tous susceptibles d'avoir des ralentissements en chaîne. Par exemple, dans Cure [3], les écritures répliquées portent des méta-données qui explicitent les dépendances causales. Le centre de données retarde ensuite l'application jusqu'à ce que ces dépendances soient appliquées localement. La visibilité d'une écriture dans un serveur peut alors devenir dépendante du délai de propagation d'une autre écriture.
La figure 1 montre un exemple simple de scénario déclenchant des ralentissements en chaine. X et Y sont deux données contenues dans le centre de données DC1, Y dépend causalement de X, or seule Y a été propagée au DC2. La requête du client, qui demande à lire Y, est donc retardée par le fait que la donnée X dont dépend Y n'a pas été propagée par le DC1.
Ce problème est encore plus important dans la pratique, puisque dans un système réel, on peut

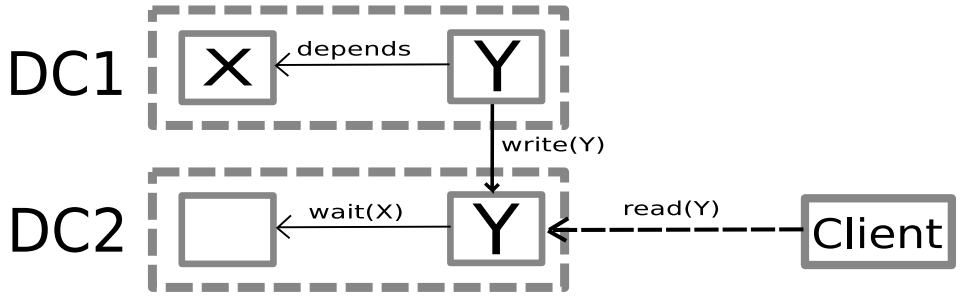

FIGURE 1 – Exemple de ralentissement en chaîne, le retard de réplication de X retarde la lecture de Y

avoir un grand nombre de requêtes de lecture, sur un grand nombre de données répliquées sur plusieurs centres de données. Et si un serveur ne peut suivre le taux d'arrivée des écritures, alors il retarde l'ensemble du système.
On peut donc dire que ces *ralentissements en chaîne* sont un handicap pour l'évolutivité de la cohérence causale.

4. Idée

La cohérence causale exige dans le cas de la réplication d'une écriture, que l'ensemble des écritures dont elle dépend soient appliquées, ce qui renforce le modèle, mais laisse le système vulnérable aux *ralentissements en chaîne*.
Pour résoudre ce problème, nous souhaitons déplacer ce choix aux clients, et ainsi laisser les clients eux-mêmes déterminer s'il est sécurisé de lire une valeur sans avoir d'abord à appliquer toutes les écritures précédemment causales, et ainsi rendre les écritures immédiatement visibles.
L'enjeu principal, cependant, est d'identifier un encodage de méta-données qui minimise à la fois la surcharge et la latence de lecture. En effet, le schéma général d'un système qui fait passer l'application de la cohérence causale aux opérations de lecture est simple : chaque client doit maintenir certaines méta-données relatives à l'état le plus récent du centre de données qu'il a observé. Lors de la lecture d'un objet X, le client doit déterminer si la version de X stockée dans le centre de données est sûre à lire (c'est-à-dire qu'elle reflète toutes les mises à jours stockées dans les méta-données du client), ainsi, si la version est jugée correcte à lire, le client doit mettre à jour ses méta-données pour appliquer toute nouvelle dépendance ; Si ce n'est pas le cas, alors le client devrait décider comment procéder (attendre ou contacter un autre centre de données). Ainsi, comme chaque objet du magasin doit être augmenté avec ses méta-données, l'importance de réduire sa taille est évidente.

5. Modèle du système

Notre système est une base de données clé-valeur géo-répliquée, où chaque réplique est située dans un data-center séparé avec une copie complète des données (2). Cette réplication complète permet de fournir disponibilité et réduire la latence. Chaque centre de données est séparé

en plusieurs partitions, chaque partition peut stocker un ensemble de plusieurs dizaines ou centaines de milliers de clés disjointes, placés sur le même hôte physique. Nous considérons que chaque centre de données utilise le même schéma de partitionnement.
Au niveau des méta-données, notre protocole repose sur les datages (*timestamps*) des événements pour encoder les dépendances causales qui permettent aux partitions de prendre des décisions localement. Cela évite l'utilisation explicite des messages de contrôle de dépendance, ce qui pénalise la performance.
Les clients de notre système sont co-situés dans le même data-center que l'une des répliques. Chaque client lit à partir de sa réplique locale et peut éventuellement écrire dans une réplique distante. L'API client applique la cohérence causale pour les lectures et les méta-données aux écritures.

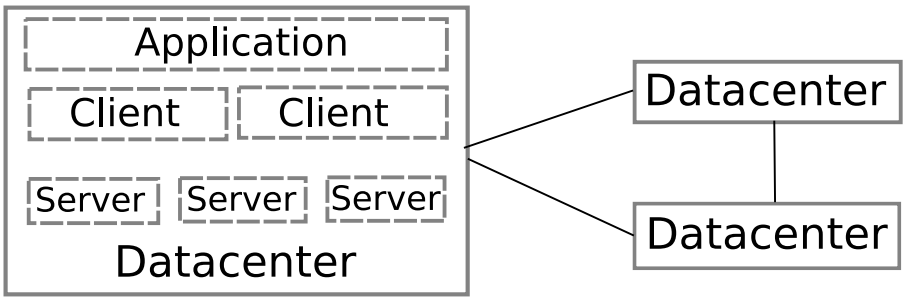

FIGURE 2 – Architecture du système.

6. Protocole de transactions

De nombreuses applications peuvent bénéficier de la capacité de lite et d'écrire plusieurs objets atomiquement. Une transaction peut donc être exécutée sur n'importe quel centre de données et peut accéder aux éléments de l'une des répliques. Une transaction est engagée localement sur un serveur, puis, à un moment ultérieur, ses changements sont propagés de manière asynchrone vers d'autres sites et y sont appliqués. Les transactions sont ordonnées selon un ordre causal. Le *snapshot* (instantané) observé par une transaction peut ne pas refléter les dernières versions des éléments consultés, mais il est garanti pour être causalement cohérent.

Notre modèle fournit les garanties suivantes :

- Isolation : si deux écritures concurrentes ont lieu sur un même objet, seule une des deux sera prise en compte. Cette garantie est cruciale pour éliminer le danger des anomalies comme les modifications perdues.
- Cohérence global d'un snapshot (instantané) : Un snapshot cohérent respecte deux propriétés :
  - Atomicité : dans un snapshot cohérent, toutes ou aucune des modifications d'une transaction sont visibles.

- Causalité : si un snapshot contient des mises à jour de la transaction Ti, les mises à jour de toutes les transactions qui précèdent causalement Ti y sont également contenues.
- Ordre des transactions : Deux transactions non concurrentes Ti et Tj sont ordonnées comme suit :
  - Ordre causal : Si Tj lit l'une des écritures effectuées par Ti, alors la transaction Ti précède causalement Tj (Ti < Tj)
  - Ordre de mise à jour global par objet : Ti < Tj si Tj crée une version plus récente pour l'un des objets modifiés par Ti, c'est à dire que Ti est pris en compte avant Tj.
  - Ordre des mises à jour par partition : Ti < Tj si Ti et Tj s'exécutent sur le même site, modifient une partition commune et Ti obtient le datage de sa modification avant Tj.
  - L'ordre des transactions est transitif, c'est à dire que si Ti < Tj et que Tj < Tk, alors Ti < Tk.
  - Ordre total des transactions : Là où PSI [16] ordonne totalement toutes les transactions qui modifient une réplique, exigeant que cet ordre soit respecté lors de la réplication sur d'autres sites, ce qui peut donc provoquer le ralentissement de certaines transactions dépendantes de celles se trouvant sur un serveur lent ou défaillant. Notre modèle supprime ces contraintes inutiles. Plutôt que d'ordonner totalement toutes les transactions qui se trouvaient sur la meme réplique, notre protocole exige seulement que les transactions soient répliquées de manière à respecter les dépendances en lecture/écriture et l'ordre des transactions qui appartiennent à la même session du client.
  - Ces contraintes sont suffisantes pour assurer des dépendances sémantiques pertinentes. Par exemple, dans le cas d'un réseau social, si Alice supprime Bob de sa liste d'amis, puis charge ensuite ses photos depuis une autre transaction, Bob ne pourra pas lire les photos quelque soit la réplique depuis laquelle il demande sa lecture.

7. Travaux connexes

Les travaux récents ont mis l'accent sur l'amélioration des conceptions AP (disponibilité sous partition) avec une sémantique plus forte [4, 14]. Ces solutions offrent une variété d'interfaces transactionnelles limitées mais intéressantes, qui visent à faciliter le développement des applications. COPS [13] a introduit le concept des transactions causalement cohérentes en "lecture-seule" (*read-only transactions*). Il trace la cohérence causale grâce aux dépendances explicites, puis l'applique de façon pessimiste en vérifiant ces dépendances avant d'appliquer les écritures distantes sur une réplique. Ce concept a été ensuite adopté par ChainReaction [13] et Orbe [8] qui utilisent des filtres de Bloom pour réduire les dépendances. Eiger [14] a étendu cette interface transactionnelle avec des transactions causalement cohérentes en "écriture-seule" (*write-only transactions*). Pour déterminer si une mise à jour peut être rendue visible, ces modèles utilisent des mécanismes qui s'appuient sur l'échanges d'informations de dépendance causale et de messages explicites de contrôle de dépendance entre les différents nœuds, ce qui provoque une augmentation de la taille des méta-données.
Plus récemment GentleRain [9] propose d'éviter ce coût de vérification des dépendances, en utilisant un algorithme de stabilisation globale pour rendre les mises à jour visibles dans les centres de données.
Le modèle que nous utilisons dans Antidote [5], Cure [3], suit ce choix de conception et réalise un débit similaire tout en offrant aux programmeurs une sémantique plus forte, à savoir des transactions en lecture-écriture. Cure utilise un timestamp unique par réplique pour suivre et

appliquer les dépendances causales. Il exige également que toutes les opérations soient sur des types de données répliqués convergents et commutatifs (CRDTs, de l'anglais, *Convergent and Commutative Replicated Data Types*) [17]. L'utilisation des CRDTs permet à Cure d'éviter la coordination des écritures en fusionnant les écritures conflictuelles, y compris celles émises dans le cadre de transactions de lecture-écriture.

8. Conclusion

Cet article identifie les ralentissements en chaîne comme une limitation fondamentale à l'application de la cohérence causale en tant que propriété globale d'une base de données géo-répliquée. Notre système délègue plutôt cette responsabilité au client, ainsi le centre de données applique les mises à jours dès qu'il les reçoit. Les clients attendent uniquement la propagation des écritures par lesquelles ils sont réellement intéressés. Le protocole de transactions proposé par notre système assure que les transactions sont lues à partir d'un snapshot cohérent tout en préservant l'atomicité, il garantit donc les fortes propriétés de PSI tout en évitant les ralentissements qui peuvent compromettre son passage à l'échelle. Notre système essaie de trouver un équilibre entre entre la surcharge des méta-données et le "faux partage", en sélectionnant dans les centres de données, les sérialisations qui minimisent les dépendances parasites.

Bibliographie

1. Abadi (D.). – Consistency tradeoffs in modern distributed database system design : Cap is only part of the story. *Computer*, vol. 45, n2, 2012, pp. 37–42.
2. Ahamad (M.), Neiger (G.), Burns (J. E.), Kohli (P.) et Hutto (P. W.). – Causal memory : Definitions, implementation, and programming. *Distributed Computing*, vol. 9, n1, 1995, pp. 37–49.
3. Akkoorath (D. D.), Tomsic (A. Z.), Bravo (M.), Li (Z.), Crain (T.), Bieniusa (A.), Preguiça (N.) et Shapiro (M.). – Cure : Strong semantics meets high availability and low latency. – In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pp. 405–414. IEEE, 2016.
4. Almeida (S.), Leitão (J.) et Rodrigues (L.). – Chainreaction : a causal+ consistent datastore based on chain replication. – In *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 85–98. ACM, 2013.
5. Antidote. – Antidote db - http ://syncfree.github.io/antidote/, 2015.
6. Birman (K.), Chockler (G.) et van Renesse (R.). – Toward a cloud computing research agenda. *ACM SIGACt News*, vol. 40, n2, 2009, pp. 68–80.
7. DeCandia (G.), Hastorun (D.), Jampani (M.), Kakulapati (G.), Lakshman (A.), Pilchin (A.), Sivasubramanian (S.), Vosshall (P.) et Vogels (W.). – Dynamo : amazon's highly available key-value store. *ACM SIGOPS operating systems review*, vol. 41, n6, 2007, pp. 205–220.
8. Du (J.), Elnikety (S.), Roy (A.) et Zwaenepoel (W.). – Orbe : Scalable causal consistency using dependency matrices and physical clocks. – In *Proceedings of the 4th annual Symposium on Cloud Computing*, p. 11. ACM, 2013.
9. Du (J.), Iorgulescu (C.), Roy (A.) et Zwaenepoel (W.). – Gentlerain : Cheap and scalable causal consistency with physical clocks. – In *Proceedings of the ACM Symposium on Cloud Computing*, pp. 1–13. ACM, 2014.
10. Gilbert (S.) et Lynch (N.). – Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, vol. 33, n2, 2002, pp. 51–59.

11. Lakshman (A.) et Malik (P.). – Cassandra : a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, vol. 44, n2, 2010, pp. 35–40.
12. Lamport (L.). – Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, vol. 21, n7, 1978, pp. 558–565.
13. Lloyd (W.), Freedman (M. J.), Kaminsky (M.) et Andersen (D. G.). – Don't settle for eventual : scalable causal consistency for wide-area storage with cops. – In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 401–416. ACM, 2011.
14. Lloyd (W.), Freedman (M. J.), Kaminsky (M.) et Andersen (D. G.). – Stronger semantics for low-latency geo-replicated storage. – In *NSDI*, pp. 313–328, 2013.
15. mongodb. – Mongodb - https ://www.mongodb.com, 2009.
16. Padhye (V.), Rajappan (G.) et Tripathi (A.). – Transaction management using causal snapshot isolation in partially replicated databases. – In *Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on*, pp. 105–114. IEEE, 2014.
17. Shapiro (M.), Preguiça (N.), Baquero (C.) et Zawirski (M.). – *A comprehensive study of convergent and commutative replicated data types*. – Thèse de PhD, Inria–Centre Paris-Rocquencourt ; INRIA, 2011.