Le partage d'informations dans les systèmes répartis grande échelle

Marc Shapiro

▶ **To cite this version:**

Marc Shapiro. Le partage d'informations dans les systèmes répartis grande échelle. René Jacquart. Informatiques, enjeux tendances et évolutions, 19, Hermès Science, pp.455-462, 2000. inria-00444643

HAL Id: inria-00444643
https://inria.hal.science/inria-00444643

Submitted on 7 Jan 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

RÉSUMÉ. Dans les systèmes informatiques répartis, le partage de l'information est assuré par la réplication. Le maintien de la cohérence entre réplicats bute sur plusieurs problèmes, en particulier l'impossibilité du consensus. Ces difficultés sont contournées par la cohérence optimiste, qui laisse diverger les réplicats pour les réconcilier a posteriori.

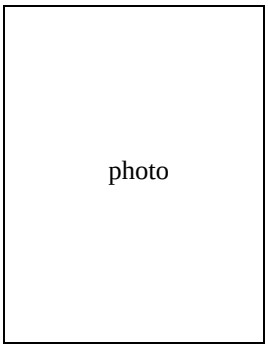
ABSTRACT. In distributed computer systems, information sharing is based on replication. Maintaining consistency between replicas has various limitations, in particular the impossibility of consensus. Optimistic concurrency algorithms are able to side-step these difficulties by letting replicas diverge, and reconciling a posteriori.

MOTS-CLÉS : systèmes répartis, réplication, caches, cohérence, réconciliation

***Mots-clé** : distributed systems, replication, caching, consistency, reconciliation*

Marc Shapiro
Microsoft Research Cambridge et INRIA projet SOR
mailto:marc.shapiro@inria.fr , http://www-sor.inria.fr/~shapiro/

Le partage d'information dans les systèmes répartis de grande échelle



Marc Shapiro, *Directeur de Recherche INRIA, est actuellement chercheur au laboratoire Microsoft Research de Cambridge (Royaume-Uni). Auparavant responsable du projet SOR de l'INRIA Rocquencourt, il est l'auteur de plusieurs systèmes répartis à objets, dont SOS et Larchant. Son sujet de recherche de prédilection est la persistance et le ramasse-miettes dans les systèmes répartis. En tant que responsable du projet européen PerDiS, il dirige la conception et la mise en œuvre d'un système de partage des données à grande échelle pour l'ingénierie coopérative.*

Objets de recherche depuis longtemps, les systèmes répartis sont devenus familiers avec la popularisation de l'Internet, qui atteint un nombre d'utilisateurs, une étendue géographique, et une diversité étonnants. Ce «passage à l'échelle» met durement à l'épreuve les algorithmes répartis.

Le but de ce bref papier est de montrer la place essentielle du partage de l'information, et donc de la réplication, dans un système réparti. Le problème central de ces systèmes est la cohérence des données, qui est relié au résultat d'«impossibilité du consensus». Afin de contourner ces problèmes, les grands services de l'Internet adoptent une structure en

0. In : Informatiques, enjeux tendances et évolutions. Sous la direction de René Jacquart. Techniques et Sciences informatiques, volume 19, numéro 1-2-3, janvier-mars 2000.

«grappes» centralisées. À l'autre extrémité du spectre, on voit apparaître des services nomades personnalisés. Ces deux configurations extrêmes ont en commun le problème de la divergence et de la réconciliation des données.

Nous commençons notre exposé, en partie 1, par un exemple familier, le service DNS. La partie 2 s'intéresse au partage des données en général, et de la nécessaire réplication. Dans la partie 3, nous exposons les problèmes inhérents à la réplication. La partie 4 explique brièvement la persistance des données. Enfin, la partie 5 expose les concepts de la réplication optimiste. La conclusion sera la partie 6.

1. Le système de nommage réparti DNS

Considérons un exemple familier et intéressant, celui du service de nommage DNS (Domain Name Service) [MOC 87]. Le DNS constitue une base de données très répartie, associant un nom de site vers l'adresse IP (numérique) correspondante. C'est le DNS qui permet à l'utilisateur de l'Internet d'employer des noms de machines, sans avoir à connaître les adresses.

Lorsque votre butineur veut accéder au site nommé www.inria.fr, il soumet ce nom au mandataire local [SHA 86] du service DNS, lequel retournera l'adresse. Si l'adresse du site www.inria.fr a été demandée récemment, le mandataire l'aura mémorisée dans un cache et pourra répondre immédiatement. Dans le cas contraire, le mandataire fait suivre la requête à un serveur DNS sur le réseau ; quand la réponse arrivera, le mandataire stocke la réponse dans son cache et la retourne au demandeur. Éventuellement, le serveur distant peut servir de mandataire pour un deuxième serveur, qui lui-même pourra jouer de rôle de cache de troisième niveau, et ainsi de suite jusqu'au serveur primaire qui fait autorité pour www.inria.fr.

L'utilisation du cache diminue le temps de réponse et diminue la charge du réseau. De plus elle évite qu'un problème de réseau temporaire ne bloque l'utilisateur dans son travail, si bien sûr la donnée est déjà dans un cache accessible.

Un serveur peut changer d'adresse (par exemple, un serveur Web fortement chargé peut reporter sa charge sur un autre) en changeant son mappage dans le DNS. Le cache devient alors incohérent par rapport au primaire. Pour limiter les incohérences, une entrée de cache aura une durée de vie limitée. Le service DNS s'accomode des incohérences, car d'une part elles sont rares, temporaires et inoffensives, d'autre part l'utilisateur les détecte et relance la requête.

Les concepteurs du DNS ont choisi des solutions qui passent à l'échelle. Ainsi, le DNS maximise l'autonomie entre sites et s'adapte au fur et à

mesure des modifications du réseau. Le DNS n'est pas strictement hiérarchique. Chaque mandataire a le choix entre plusieurs serveurs, un grand nombre de serveurs primaires existent, et un même nom peut être traduit par plusieurs serveurs primaires. Ces choix sont heureux, car le bon fonctionnement de l'Internet repose en grande partie sur celui du DNS.

2. Le partage des données et la réplication

Dans un système réparti, les mémoires des différents sites sont disjointes. Malgré (ou à cause de) cela, de nombreux algorithmes répartis sont soit utilisateurs, soit fournisseurs d'une abstraction de mémoire partagée. Ainsi le DNS offre une mémoire partagée des mappages entre nom et adresse.

Dans une architecture client-serveur, la mémoire commune est localisée sur un ou plusieurs sites serveurs. Elle n'est pas directement visible par les sites clients, mais seulement par l'intermédiaire d'une interface procédurale. Les programmes applicatifs, s'exécutant sur des sites clients, exécutent les procédures par échange de messages avec le serveur. Ce schéma, simple à comprendre et à mettre en œuvre, est celui offert par des systèmes commerciaux comme Corba ou DCOM. Son défaut est qu'un message coûte jusqu'à plusieurs centaines de milliers de cycles CPU, à comparer aux quelques cycles d'un accès mémoire local. Quand les clients accèdent fréquemment à la mémoire commune, le serveur devient goulot d'étranglement, et le coût de communication devient prohibitif : le client-serveur ne passe pas à l'échelle.

Dans le paradigme des agents mobiles, la mémoire partagée est l'union des mémoires locales. C'est le programme applicatif lui-même qui se déplace vers les données dont il a besoin. Ce schéma est attrayant dans le cas où la taille du programme mobile est plus faible que le volume de données partagées. L'exemple souvent cité est une application de commerce électronique, où un agent acheteur recherche le meilleur prix dans les bases de données de plusieurs vendeurs. Cette technologie n'est pas encore au point, pour des problèmes de mise en œuvre, de tolérance aux pannes et de langage.

Puisque l'accès distant est si coûteux, pourquoi ne pas maintenir une copie, dit réplicat [1], de la donnée sur le site demandeur ? Après le premier transfert, coûteux, les lectures suivantes de la même donnée seront locales. Dans le même message on transférera tout un bloc, chargeant le cache avec la donnée demandée et les données environnantes. Le cache est donc bénéfique si les applications ont les propriétés de localité temporelle (une donnée accédée tend à être rapidement accédée à nouveau)

1. Nous ne ferons pas la distinction ici entre cache et réplicat, qui est d'ordre technique.

et spatiale (le programme tend à accéder aussi les données voisines). Empiriquement, de très nombreux systèmes présentent ces deux propriétés.

Les systèmes dits de mémoire partagée répartie (Distributed Shared Memory ou DSM) donnent l'accès aux réplicats locaux à travers une interface de mémoire virtuelle. Le modèle est le même qu'une mémoire centralisée ; leur intégration dans des langages de bas niveau comme C ou C++ est immédiate. Grâce au cache, leur performance peut être très bonne. Ces avantages ont été démontrés dans le système PerDiS [FER 98b], où l'utilisation d'une interface mémoire a permis de rendre réparties relativement facilement de grosses applications centralisées préexistantes.

3. Les limites de la réplication

Le cas de l'écriture est défavorable à la réplication, puisqu'il faut alors gérer la cohérence entre réplicats. La sémantique de la cohérence peut être plus ou moins forte [THI 97]. La plus intuitive est la cohérence dite atomique, mais son coût est élevé, et elle impose un couplage fort (donc coûteux et fragile) entre sites. Une cohérence aussi faible que celle DNS n'impose pas un tel couplage, mais donne en contrepartie des garanties insuffisantes pour être utilisées par un programme d'une certaine complexité (par exemple qui doit à la fois lire et écrire plusieurs valeurs).

Tout système réparti est donc confronté au douloureux compromis entre cohérence et performance, et entre autonomie des sites et simplicité du programme.

Les pannes apportent une difficulté supplémentaire. Dans l'Internet on observe aussi bien des pannes de site que de message. Or la détection de pannes par temporisation sur les messages n'est pas fiable car on ne peut pas borner le temps de livraison d'un message (l'Internet est un système asynchrone).

Examinons un instant le problème de la cohérence sans rentrer dans les détails d'un algorithme de cohérence particulier. Plusieurs sites détiennent leur réplicat d'une variable x ; chacun peut lire et écrire sa copie locale[2]. À un certain moment, les sites doivent se mettre d'accord sur la valeur courante de x. Chaque site en propose une ; la cohérence doit choisir une valeur commune parmi celles proposées. C'est ce qu'on appelle le «problème du consensus». Or d'après le théorème célèbre de Fisher, Lynch et Patterson [FIS 85], le consensus n'est pas possible dans un système asynchrone en présence de pannes. Ceci, même sur la valeur d'un seul bit, et quelle que soit la loi (déterministe) de choix entre propositions. À fortiori, aucun algorithme déterministe de cohérence ne peut

2. Pour être complètement général, ignorons l'ordonnancement entre lectures et écritures, éventuellement imposé par un contrôle de concurrence associé à l'algorithme de cohérence.

fonctionner en présence de pannes, et celles-ci sont inévitables dans un système de grande échelle.

4. La persistance et le ramasse-miettes

En plus du partage entre processus concurrents, il faut penser aux processus s'exécutant à des moments différents. Pour cela, la mémoire partagée sera persistante, c'est-à-dire qu'il existe un réplicat supplémentaire sur un support stable, comme un disque. Une donnée peut être répliquée sur plusieurs disques géographiquement dispersées, afin de les rendre plus disponibles, en particulier en cas de panne. Or un disque est souvent plus lointain, en termes de coût de communication, qu'un site distant, ce qui exacerbe les problèmes de cohérence. En cas de panne ou de déconnexion prolongée d'un site persistant, le problème de divergence se posera de façon aigüe.

Le modèle le plus naturel est la persistance par accessibilité [ATK 83], dans lequel tout objet est persistant dès lors qu'il est accessible. C'est celui mis en œuvre dans notre projet européen PerDiS, qui offre une mémoire persistante répartie conçue pour des applications de CAO coopératives [Pro 99]. Un ramasse-miettes détruira les objets devenus inaccessibles. Le ramasse-miettes réparti pose des problèmes de recherche passionnants, surtout dans le contexte de la réplication [FER 98a].

Le projet Intermemory [GOL 98] constitue une mémoire persistante originale. Chaque utilisateur fait don d'une certaine quantité de disque dur à la communauté Intermemory. En échange le système promet de conserver sa mémoire de façon particulièrement sûre et fiable. Tout fichier est crypté et découpé, les morceaux étant massivement répliqués, à des emplacements aléatoires. Un fichier Intermemory devient à la fois inviolable et indestructible. D'après les auteurs, le ramasse-miettes n'est pas nécessaire, car l'espace disque disponible sur l'Internet croît plus vite que les besoins.

5. La réplication optimiste

L'Internet est le théâtre de changements profonds et rapides. Les changements les plus spectaculaires apparaissent aux deux extrémités de l'échelle. Dans les deux cas, les programmeurs d'application sont directement confrontés aux problèmes théoriques exposés ci-dessus.

D'une part des machines toujours plus petites, portables et spécialisées se multiplient comme le Palm Pilot ou les cartes à puce. Les données comme les agendas ou les boîtes à lettres électroniques sont naturellement répliquées sur une machine nomade et une (ou plusieurs) machines fixes.

Ces machines nomades ne sont connectées au réseau que de façon épisodique ; le délai de livraison d'un message peut atteindre des jours ou des semaines. Les réplicats ayant pu diverger pendant ce temps, il faut les réconcilier à posteriori. Cette réplication est dite «optimiste» car basée sur l'hypothèse que les divergences sont rares et solubles.

À l'autre extrémité du spectre, des services comme les portails, les serveurs de courrier, les sites de commerce électronique, servent des millions de requêtes par jour. Les services de HotMail ou de Yahoo s'exécutent chacun sur près d'un millier de machines simultanément ! Une donnée sera répliquée sur un grand nombre de ces machines afin d'augmenter le parallélisme. La tendance est de concentrer tous les serveurs d'un service dans la même pièce (formant des «grappes» ou clusters de machines), avec des connexions réseau de très gros débit vers les points d'utilisation éventuellement lointains. Ces grands services sont souvent basés sur une réplication optimiste, qui permet un meilleur parallélisme, donc de meilleures performances. Ici encore le problème de la réconciliation se posera.

La réconciliation constitue un champ de recherche encore peu exploré. Si les mises à jour sont commutatives, on peut réconcilier automatiquement en les ré-appliquant dans un ordre quelconque. Dans le cas d'une boîte à lettres ou d'un agenda, où l'information est presque toujours ajoutée, jamais enlevée, la réconciliation entre deux réplicats divergents se résume à faire leur union.

Bayou [PET 97] propose une technique générale, où chacun des réplicats maintient un journal des modifications locales. Lorsque deux réplicats se rencontrent, ils cherchent à fusionner leurs journaux selon un ordonnancement acceptable ; la notion d'acceptabilité dépend du type de données. Si un tel ordre est trouvé, les opérations dans le journal sont réexécutées à partir du dernier état commun. Si aucun ordre commun ne peut être trouvé, le système appelle une procédure de réconciliation fournie par le programmeur.

Les divers «synchroniseurs» commerciaux de fichiers ou d'agendas utilisent des techniques de détection de conflit et de reconciliation très primitives. Le système Lotus Notes détecte les conflits mais la réconciliation reste manuelle. La recherche dans ce domaine en est encore à ses débuts.

6. Conclusion

Bien des aspects importants ont été passés sous silence. Ainsi nous n'avons quasiment rien dit des grandes applications comme le Web, le commerce électronique, etc. Dans un système réparti ouvert comme l'Internet, se posent d'importants problèmes de sécurité et de standardisation.

Il y aurait sans doute beaucoup à dire sur les paradigmes de programmation et sur les langages. Les performances et la programmabilité du système constituent des sujets de recherche inépuisables.

Nous avons préféré nous concentrer sur le défi fondamental : le partage des données, donc leur réplication, donc la cohérence et la gestion des réplicats. Il faut en retenir l'impossibilité du consensus et l'absence d'algorithmes de cohérence universellement satisfaisants. La «transparence» du réseau, c'est-à-dire le fait que l'utilisateur ne puisse pas distinguer pas le comportement du système réparti de celui d'un système centralisé, est donc sans doute un mythe inaccessible à grande échelle. Les imperfections du réseau se manifestent par une divergence entre réplicats ; ce qui à son tour pose le problème de la réconciliation.

Ces difficultés conduiront les programmeurs à s'assigner des objectifs et une échelle raisonnables à leurs systèmes répartis. Elles n'arrêtent pas leur créativité ; les systèmes répartis fonctionnent, nous le voyons tous les jours.

Bibliographie

[ATK 83] ATKINSON M. P., BAILEY P. J., CHISHOLM K. J., COCKSHOTT P. W. et MORRISON R., « An Approach to Persistent Programming ». *The Computer Journal*, vol. 26, n° 4, p. 360–365, 1983.

[FER 98a] FERREIRA P. et SHAPIRO M., « Modelling a Distributed Cached Store for Garbage Collection ». In *12th Euro. Conf. on Object-Oriented Prog. (ECOOP)*, Brussels (Belgium), July 1998. http://www-sor.inria.fr/publi/MDCSGC_ecoop98.html.

[FER 98b] FERREIRA P., SHAPIRO M., BLONDEL X., FAMBON O., GARCIA J., KLOOSTERMAN S., RICHER N., ROBERTS M., SANDAKLY F., COULOURIS G., DOLLIMORE J., GUEDES P., HAGIMONT D. et KRAKOWIAK S., « PerDiS: design, implementation, and use of a PERsistent DIstributed Store ». Rapport technique QMW TR 752, CSTB ILC/98-1392, INRIA RR 3525, INESC RT/5/98, QMW, CSTB, INRIA and INESC, October 1998. http://www-sor.inria.fr/publi/PDIUPDS_rr3525.html.

[FIS 85] FISHER M., LYNCH N. et PATTERSON M., « Impossibility of distributed consensus with one faulty process ». *Journal of the ACM*, vol. 32, n° 2, p. 274–382, April 1985.

[GOL 98] GOLDBERG A. V. et YIANILOS P. N., « Towards an Archival Intermemory ». In *Proc. IEEE International Forum on Research and Technology Advances in Digital Libraries (ADL'98)*, p. 147–156. IEEE Computer Society, April 1998. http://www.intermemory.org/papers/intermemory/main.html.

[MOC 87] MOCKAPETRIS P.. « Domain names — concepts and facilities ». ftp://ftp.inria.fr/rfc/rfc10xx/rfc1034.Z, November 1987.

[PET 97] PETERSEN K., SPREITZER M. J., TERRY D. B., THEIMER M. M., et DEMERS A. J., « Flexible Update Propagation for Weakly Consistent Replication ». In *Proc. Symp. on Operating Systems Principles (SOSP-16)*, p. 288–301, Saint Malo, October 1997. ACM SIGOPS. http://www.parc.xerox.com/csl/projects/bayou/.

[Pro 99] PROJET EUROPÉEN PERDIS. « site Web ». http://www.perdis.esprit.ec.org/, 1999.

[SHA 86] SHAPIRO M., « Structure and Encapsulation in Distributed Systems: the Proxy Principle ». In *The 6th International Conference on Distributed Computer Systems*, p. 198–204, Cambridge, Massachusetts, May 1986. IEEE.

[THI 97] THIA-KIME G., « *Critères de cohérence pour données partagées à support réparti* ». thèse de doctorat, Université de Rennes 1, October 1997. ftp://ftp.irisa.fr/techreports/theses/1997/thiakime.ps.gz.