

Extension du modèle de cohérence à l'entrée pour la visualisation dans les applications de couplage de codes sur grilles

Loïc Cudennec, Sébastien Monnet
IRISA/Université de Rennes 1
Campus de Beaulieu, 35042 Rennes, France
Contact : {Loic.Cudennec|Sebastien.Monnet}@irisa.fr

Résumé

Ce papier s'intéresse au problème de la visualisation des données partagées dans les applications à base de couplage de codes sur les grilles. Nous proposons d'améliorer l'efficacité de la visualisation en intervenant sur les mécanismes de gestion des données répliquées et plus particulièrement au niveau du protocole de cohérence. La notion de *lecture relâchée* est alors introduite comme une extension du modèle de cohérence à l'entrée (*entry consistency*). Ce nouveau type d'opération peut être réalisé sans prise de verrou, en parallèle avec des écritures. En revanche, l'utilisateur *relâche* les contraintes sur la *fraîcheur* de la donnée et accepte de lire des versions *légèrement* anciennes, dont le retard est néanmoins contrôlé. L'implémentation de cette approche au sein du service de partage de données pour grilles JuxMem montre des gains considérables par rapport à une implémentation classique basée sur des lectures avec prise de verrou.

1 Introduction

Le problème de la cohérence des données en environnement réparti a fait l'objet de nombreuses études, notamment dans le contexte du calcul parallèle à hautes performances pour des machines multiprocesseurs à mémoire partagée, dans le domaine des systèmes à mémoire virtuellement partagée (MVP), etc. Ce problème intervient pour tout système proposant de fournir l'abstraction d'un espace global de données au dessus d'une infrastructure physiquement répartie. Il reste d'actualité aujourd'hui, lorsque des infrastructures de calcul et de stockage réparties, telles que les grilles et les systèmes pair-à-pair, deviennent disponibles à des échelles de plus en plus larges. Néanmoins, la plupart des modèles et protocoles de cohérence traditionnels font l'hypothèse d'une infrastructure *statique, sans fautes*. Ces hypothèses ne sont plus valides à l'échelle des grilles de calcul. La *tolérance aux fautes* et la prise en compte de la *volatilité des ressources* deviennent indispensables et nécessitent une remise en cause des techniques traditionnelles. La prédominance des systèmes basés sur les transferts *explicites* de données au sein des grilles actuelles (GridFTP [3], IBP [6], etc.) montre que le partage transparent de données à grande échelle reste encore un défi.

La plate-forme JuxMem [4] aborde ce problème en proposant le concept de *service de partage transparent de données*, qui vise essentiellement les applications de calcul sur grille. JuxMem suit une approche hybride, inspirée par les systèmes à mémoire virtuellement partagée (MVP) d'une part et par les systèmes pair-à-pair d'autre part. Basée sur la librairie pair-à-pair JXTA [1], JuxMem est une plate-forme d'expérimentation de protocoles de cohérence tolérants aux fautes qui intègre des mécanismes permettant la gestion des groupes de pairs, la

diffusion atomique et le consensus, en présence de fautes. Ces blocs de base peuvent être utilisés directement par les protocoles de cohérence pour renforcer la disponibilité des entités critiques telles que les gestionnaires, les noeuds-hôtes (*home nodes*), etc. Cette approche détaillée dans [5] a été illustrée par un protocole de cohérence hiérarchique, qui implémente le modèle de cohérence à l'entrée, (*entry consistency*). D'autres protocoles peuvent être mis en place au sein de la plate-forme JuxMem pour répondre à des besoins applicatifs particuliers.

Une classe d'applications portée sur les grilles de calculateurs est basée sur le *couplage de codes* : un calcul global est réalisé à l'aide de plusieurs codes s'exécutant en parallèle sur des grappes de calculateurs différentes. Même si la distribution des codes est effectuée de manière à minimiser les échanges de messages entre les grappes, des données peuvent néanmoins être partagées entre les différents sites. Le couplage de code est utilisé dans le calcul scientifique haute performance. Ces calculs peuvent durer des jours, voir des mois et il n'est pas souhaitable d'en attendre la fin pour vérifier que tout s'est bien passé. Des données peuvent être partagées entre les codes afin de renseigner sur l'état d'avancement du calcul. L'observation d'un calcul doit s'effectuer avec des temps d'accès courts tout en causant le moins d'interactions possibles avec les autres sites et ainsi minimiser les perturbations.

Dans ce travail, nous nous intéressons à l'observation des données intermédiaires dans une application à base de couplage de codes. L'objectif est d'améliorer le comportement du protocole de cohérence lorsque des observateurs sont utilisés. La contribution principale de ce papier est de proposer une extension du protocole de cohérence qui introduit la possibilité d'effectuer des lectures en parallèle d'une écriture. Ceci est possible si l'observateur accepte d'exploiter des données dont la version est considérée comme ancienne. Cette extension du protocole a pour conséquence une extension du modèle de cohérence utilisé. Une implémentation de cette stratégie a été effectuée dans le cadre de la plate-forme JuxMem. Des évaluations de performances ont été menées pour montrer que cette solution améliore la vitesse d'accès des observateurs sans dégrader celle des autres sites.

2 Un protocole de cohérence hiérarchique tolérant aux fautes

L'architecture de JuxMem [4] reflète l'architecture matérielle de la grille, à savoir un modèle hiérarchique composé de grappes de calculateurs interconnectées par des réseaux. Le système associe à chaque bloc de données stocké un groupe de données : c'est le *groupe de données global (global data group (GDG))*. Ce groupe est formé d'un ensemble de fournisseurs hébergeant une copie du bloc. Les fournisseurs peuvent être choisis dans plusieurs grappes et l'ensemble des fournisseurs d'un bloc situés dans une même grappe forment un *groupe de données local (local data group (LDG))*. Les clients accèdent à la donnée partagée par l'intermédiaire du LDG auquel ils sont attachés (figure 1).

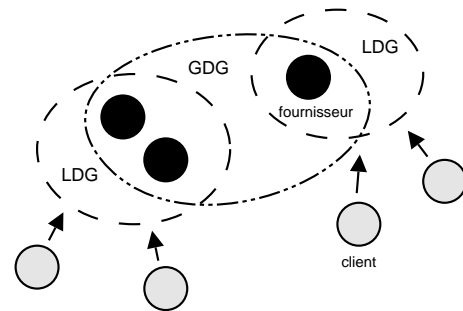


FIG. 1 – Architecture hiérarchique de JuxMem.

JuxMem intègre un protocole de cohérence hiérarchique tolérant aux fautes qui permet de mettre en œuvre le modèle de cohérence à l'entrée. Ce modèle nécessite l'association d'un objet de synchronisation (verrou) à chaque donnée partagée. Il existe 2 types de verrous : l'un en écriture garantissant l'accès exclusif à la donnée, et l'autre en lecture permettant des

lectures parallèles. Le principe du protocole utilisé par JuxMem repose sur le fait que si un client détient un verrou alors son LDG détient ce même verrou.

3 Autoriser les accès concurrents entre lectures et écritures

3.1 Proposition d'amélioration : la lecture relâchée

Le scénario considéré met en œuvre un site observateur dont le rôle est de lire la donnée partagée avec des temps d'accès courts sans pour autant dégrader les performances des autres sites. La première idée présentée dans ce papier consiste à utiliser des copies de la donnée considérées comme anciennes et détenues par le client ou son LDG. Cette stratégie permet d'exploiter des données déjà présentes sur le site client ou très proches en terme de distance réseau. La deuxième idée vise à diminuer les temps d'attente liés à la contention imposée par le protocole de cohérence en ne prenant plus le verrou en lecture. Une conséquence directe est d'autoriser ces lectures d'un genre particulier en même temps que les écritures. Ces lectures sont nommées *lectures relâchées* et la primitive associée est *rlxRead*. Le modèle de cohérence à l'entrée ne garantit qu'une donnée soit à jour que lors de l'acquisition de son verrou. Dans ce modèle, pour un observateur qui n'acquiert pas le verrou, le fait d'utiliser la donnée contenue dans son propre cache ou celle disponible sur son LDG ne peut garantir que la donnée soit à jour. L'approche que nous considérons propose d'autoriser ce type de lectures non-synchronisées tout en contrôlant la *fraîcheur* de la donnée. Ceci est possible en bornant l'écart entre la version retournée et la version la plus récente. Ainsi pour chaque lecture relâchée, l'application spécifie le nombre de versions de retard autorisées avant qu'une donnée ne soit considérée comme périmée.

3.2 Fenêtre de lecture

Pour exprimer le retard entre la version la plus récente et celle retournée par la lecture relâchée, nous introduisons deux paramètres qui prennent en compte les deux niveaux de hiérarchie du protocole de cohérence :

- La constante D , propre à chaque donnée, exprime le nombre de fois que le LDG peut transmettre successivement le verrou en écriture, sans effectuer de mise à jour du GDG. Si $D = 0$ alors le LDG doit propager les modifications après chaque relâchement du verrou en écriture. Dans ce cas tous les LDG ont la même version de la donnée.
- Le paramètre w est spécifié par le client lors de chaque appel à la primitive de lecture relâchée *rlxRead*. C'est la *fenêtre de lecture*. Elle exprime l'écart total maximum entre la version la plus récente et celle retournée par la lecture relâchée.

Les distances D et w sont positives ou nulles et respectent le fait que w soit supérieur ou égal à D . La différence $w - D$ correspond à la distance maximale entre la version détenue par le LDG du client et celle retournée par la lecture relâchée. A titre d'exemple, si $D = 3$ alors un LDG pourra distribuer successivement le verrou en écriture au plus 3 fois sans propager les modifications à tous les membres du GDG. Si $w = 4$, la version de la donnée lue par un client effectuant une lecture relâchée retourne soit la *dernière* version de la donnée détenue par son LDG, soit la version précédente. La notion de *dernière* version fait référence à la dernière version propagée par le LDG ayant le verrou en écriture. C'est à dire qu'elle peut déjà être ancienne de D versions.

3.3 Discussion sur la sémantique des paramètres

Considérer $w = D$ implique que le client lise la même version que celle détenue par son LDG. Notons que fixer $D = 0$ et $w = 0$ ne revient pas pour autant à effectuer une lecture avec prise du verrou et ce, pour deux raisons : 1) lors de la lecture relâchée, le verrou en écriture peut de nouveau être distribué et la donnée être en cours de modification. Ceci n'est pas autorisé dans le modèle de cohérence à l'entrée. 2) Entre le moment où le LDG répond à une requête de lecture relâchée et celui où le client utilise effectivement la donnée, un nombre non contrôlé de versions ont pu être produites. Cette approche convient à des applications de type visualisation mais n'offre pas de garanties strictes sur la fraîcheur des données.

Lorsqu'un LDG (dont la version de la donnée est notée V_{LDG}) reçoit une demande de lecture relâchée (de fenêtre w) de la part d'un client C (dont la version de la donnée est V_c), la formule permettant de décider si le client peut utiliser sa version de la donnée (V_c) est : $V_c \geq V_{LDG} - (w - D)$. L'utilisation de la donnée détenue par le client réduit le coût réseau de la lecture. Dans le cas contraire, la donnée du LDG est transférée au client. La lecture relâchée donne lieu à une extension du modèle de cohérence : le modèle de cohérence à l'entrée y est toujours respecté et garantit d'accéder à la version la plus récente de la donnée lorsque le verrou est acquis. La lecture relâchée apporte des garanties supplémentaires là où la cohérence à l'entrée n'en donne pas. Ainsi la lecture d'une valeur sans la prise du verrou n'est-elle plus complètement incontrôlée.

4 Evaluation préliminaire

4.1 Le scénario de l'observateur

Le programme développé pour l'évaluation des performances est un programme appartenant à la couche utilisateur et reposant sur la librairie JuxMem. Il met en jeu les fonctionnalités du protocole de cohérence dans le cadre d'un scénario particulier. Une donnée partagée est accédée par 3 clients supportant les rôles d'écrivain, lecteur et observateur. Chaque LDG est composé d'un fournisseur auquel se rattache un ou plusieurs clients. L'écrivain et le lecteur sont associés au premier LDG alors que l'observateur est attaché au second. L'écrivain effectue l'allocation de la donnée puis 50 itérations correspondant à la prise du verrou en écriture, l'écriture et le relâchement du verrou. Le lecteur effectue 50 itérations correspondant à la prise du verrou en lecture, la lecture et le relâchement du verrou. L'observateur (v_1) se comporte comme le lecteur. L'observateur (v_2) effectue 50 itérations correspondant à une lecture relâchée (*rlxRead*), sa fenêtre de lecture est $D = 0$ et $w = 0$. L'observateur (v'_2) est un observateur v_2 avec $D = 3$ et $w = 6$. La mesure des performances s'effectue en prenant l'écart entre les temps système de début et de fin d'itération dans le code des clients. On considère que les horloges des nœuds sont suffisamment synchronisées en comparaison des mesures effectuées pour présenter les temps d'accès des différents sites sur un même axe temporel.

La grappe utilisée pour les expérimentations fait partie de *Grid'5000* [2] et est composée de 64 nœuds bi-processeurs AMD Opteron cadencés à 2.2GHz, pourvus de 2Go de mémoire vive (RAM) et interconnectés par un réseau ethernet gigabit. Le développement du protocole de cohérence a été réalisé avec la version de JuxMem basée sur JXTA 2.3.2 pour J2SE (JVM 1.4.2 Sun Microsystems).

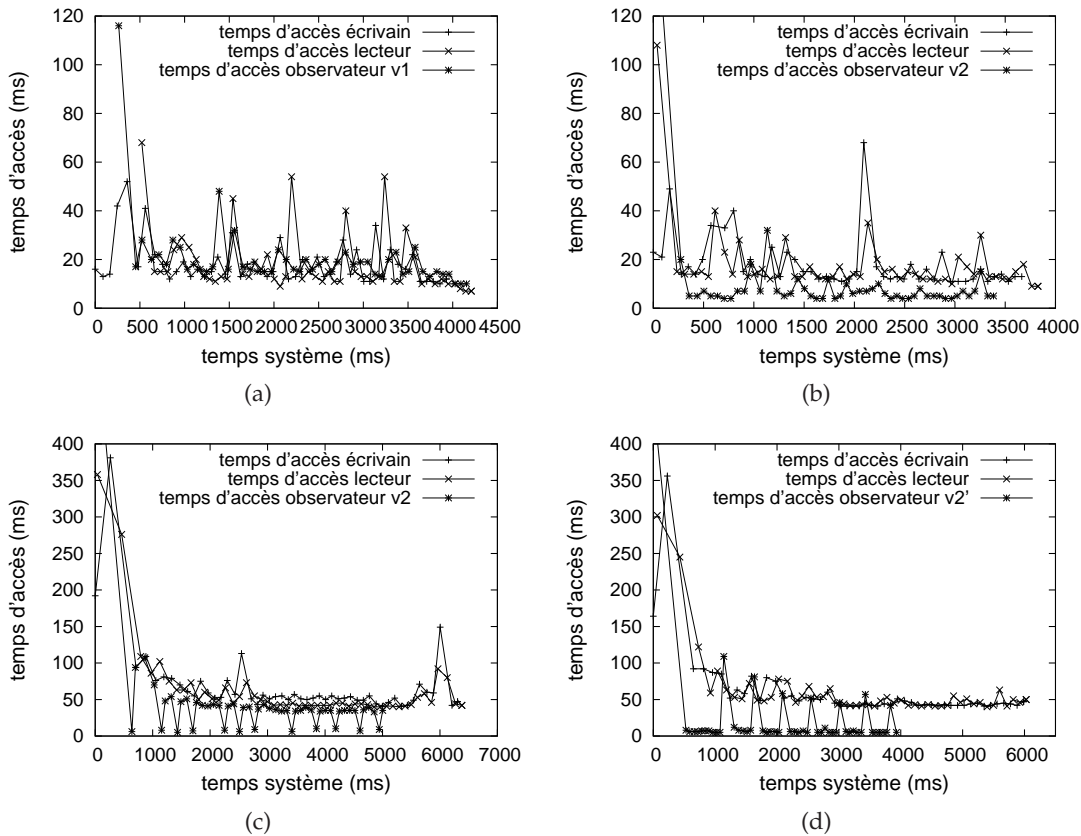


FIG. 2 – Temps d'accès à une donnée partagée. (a) v_1 , 1 ko. (b) v_2 , 1ko, $D = 0$, $w = 0$. (c) v_2 , 1024 ko, $D = 0$, $w = 0$. (d) v'_2 , 1024 ko, $D = 3$, $w = 6$.

4.2 Analyse des résultats

La séquence classique de lecture de la donnée partagée est effectuée par l'observateur v_1 (figure 2(a)). Les temps moyens d'accès de l'écrivain, du lecteur et de v_1 sont identiques (19 ms). Ceci s'explique par le fait que 1) le protocole de cohérence alterne entre la prise de verrou en écriture et en lecture et 2) le modèle de cohérence à l'entrée autorise les lectures en parallèle, ce qui profite au lecteur et à v_1 . L'utilisation de la primitive d'accès *rlxRead* par l'observateur v_2 (figure 2(b)) permet d'obtenir un temps d'observation moyen 2 fois plus petit que celui de v_1 . Ce gain de temps ne s'effectue pas pour autant au détriment des performances des autres sites. Ceci est possible par la diminution du nombre de messages générés pour une itération et la possibilité de lire en parallèle d'une écriture, sans prendre de verrou. Augmenter la taille de la donnée met en valeur les accès utilisant la copie de la donnée détenue par le client. Sur la figure 2(c), les creux intervenant sur la courbe de v_2 indiquent que la donnée n'a pas été transférée sur le réseau, celle-ci étant identique sur le client et son LDG. Les plateaux correspondent au transfert de la donnée et restent inférieurs aux temps d'accès des 2 autres sites du fait de la lecture relâchée. La figure 2(d) montre une utilisation plus fréquente de la copie de la donnée détenue par l'observateur v'_2 en augmentant la taille de la fenêtre de lecture. Le LDG de v'_2 peut détenir une copie de la donnée ayant jusqu'à 3 versions d'avance sur celle de v'_2 . Les pics correspondent alors au transfert de la donnée

devenue obsolète sur l'observateur.

5 Conclusion

L'étude du protocole de cohérence hiérarchique tolérant aux fautes de JuxMem a permis de proposer une amélioration face au scénario de l'observateur : autoriser les lectures en parallèle d'une écriture. Ces lectures, nommées *lectures relâchées* (*rlxRead*), ne sont possibles qu'en diminuant les contraintes sur la version de la donnée retournée. L'utilisateur accepte alors de lire une version de la donnée *légèrement* ancienne. Il doit spécifier, en paramètre de chacun de ses accès relâchés, le retard maximum autorisé entre la version retournée et la version la plus récente de la donnée. Pour traduire cette idée, nous proposons une extension du modèle de cohérence. Cette extension ne remet nullement en cause le modèle de cohérence à l'entrée. Les accès aux données effectués en prenant le verrou garantissent toujours de récupérer la version la plus récente. Les lectures relâchées permettent alors de proposer des garanties sur une lecture effectuée sans la prise du verrou. Les évaluations de l'extension du protocole de cohérence ont montré que les lectures relâchées permettaient un gain de temps considérable par rapport à une lecture avec prise du verrou. Par ailleurs ce gain de performance ne se réalise pas au détriment des temps d'accès des sites utilisant les primitives d'accès classiques. De nombreux points restent cependant à explorer comme l'évaluation de l'aspect dynamique de la fenêtre de lecture pendant une exécution en fonction de la charge du réseau ou de la qualité de visualisation souhaitée. Cette contribution est adaptée à des scénarios de visualisation comme la consultation de données indexées par un moteur de recherche.

Références

- [1] The JXTA (juxtapose) project. <http://www.jxta.org>.
- [2] Projet Grid'5000. <http://www.grid5000.org>.
- [3] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Comput.*, 28(5):749–771, 2002.
- [4] Gabriel Antoniu, Luc Bougé, and Mathieu Jan. JuxMem: An adaptive supportive platform for data sharing on the grid. In *Proceedings Workshop on Adaptive Grid Middleware (AGRIDM 2003)*, pages 49–59, New Orleans, Louisiana, September 2003. Held in conjunction with PACT 2003. Extended version to appear in *Kluwer Journal of Supercomputing*.
- [5] Gabriel Antoniu, Jean-François Deverge, and Sébastien Monnet. How to bring together fault tolerance and data consistency to enable grid data sharing. Technical Report PI 1685, IRISA/INRIA, University of Rennes 1, jan. 2005. Extended version to appear in *CCPE 2005*.
- [6] Alessandro Bassi, Micah Beck, Graham Fagg, Terry Moore, James S. Plank, Martin Swamy, and Rich Wolski. The internet backplane protocol: A study in resource sharing. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 194, Washington, DC, USA, 2002. IEEE Computer Society.