Un service de gestion de données persistantes partagées

P. Dechamboux, D. Hagimont, J. Mossière, X. Rousset de Pina

Laboratoire IMAG-LSR - Projet SIRAC
BP 53, 38041 Grenoble Cedex 9
email: Daniel.Hagimont@imag.fr

1 Introduction

Nous nous proposons de réaliser sur un réseau local de machines homogènes un multiprocesseur à mémoire virtuelle commune. Cette mémoire commune sera utilisée pour stocker des données partagées et donc persistantes. Les applications visées sont le support des serveurs d'information et les applications coopératives.

Trois types de problèmes se posent pour gérer ces données :

- Comment réaliser le partage et garantir la cohérence des données partagées.
- Comment résister aux pannes, en assurant également la permanence des données.
- Comment protéger les données partagées.

Les services que nous proposons prennent donc en compte ces trois types de problèmes. Cependant, nous choisissons de donner aux utilisateurs la liberté de rendre permanentes les données persistantes partagées qu'ils utilisent et de protéger l'accès à ces données. Cela impose donc de fournir et de mettre en œuvre les services de permanence et de protection de manière indépendante.
Un second choix est de ne pas imposer de politique aux utilisateurs des services mais plutôt de fournir des mécanismes permettant à l'utilisateur de mettre en œuvre la politique qui correspond le mieux aux besoins de son application.
Nous présentons dans les sections suivantes la gestion de la mémoire, de la permanence et de la protection des données (sections 2, 3 et 4). Après une brève comparaison avec des travaux similaires (section 5), nous concluons en section 6.

2 Gestion de la mémoire

Le service de partage de données comprend deux parties, une partie gestion de la mémoire partagée (allocation, destruction) et une partie gestion de la cohérence.

2.1 Service de partage des données

Notre service de mémoire partagée gère un espace virtuel unique. Les allocations de mémoire partagée se font dans un bloc de 263 octets réservés à la même adresse virtuelle dans la mémoire de tous les processus Unix qui l'utilisent. Ainsi, toute donnée partagée est vue à la même adresse virtuelle par tous les processus Unix, ce qui simplifie le partage de ces données.

L'unité d'allocation de cette mémoire est le segment qui est une suite de pages contigües. Les segments sont gérés de manière persistante. Un segment est désigné par son adresse qui reste inchangée pendant toute sa durée de vie.

À l'exécution, la gestion de la mémoire partagée repose sur les mécanismes de pagination des machines. En raison du partage des segments, des accès concurrents à une même page peuvent être effectués sur plusieurs sites simultanément, cette page étant dupliquée dans la mémoire de chacun des sites. Le système ne gère pas la cohérence entre les différentes copies d'une même page mais il fournit des mécanismes permettant aux applications de mettre en œuvre leur propre politique de cohérence.

2.2 Gestion de la cohérence

Les mécanismes offerts reposent sur les constatations suivantes :
Le segment et même la page sont des unités de grain trop gros pour être choisies comme unité de contrôle d'accès (en terme de synchronisation) et de cohérence. Il est donc nécessaire de fournir une unité de grain plus fin si on veut éviter les problèmes dus au faux partage (processus accédant à des zones disjointes de la même page).
Les applications partageant des données potentiellement accessibles simultanément par plusieurs processus synchronisent leurs accès à ces données. Il est donc possible de lier la synchronisation et la mise en cohérence des données et notamment de retarder l'application du protocole de mise en cohérence d'une zone de données lors de la demande de verrou sur cette zone.
Les mécanismes de mise en cohérence gérent des zones qui sont des suites d'octets contigus à l'intérieur des segments. Toute zone a une copie particulière dite copie maîtresse. À tout instant le système sait localiser la copie maîtresse de la zone et fournit des primitives permettant de mettre à jour une copie de zone à partir de sa copie maîtresse, ou de changer le site de résidence de la copie maîtresse d'une zone. Ce jeu de primitives permet de mettre en œuvre les politiques classiques de gestion de cohérence [4] ou de construire un protocole adapté à une application particulière.

3 Service de permanence

Dans notre système, nous distinguons donc deux niveaux de mémoire : la mémoire d'exécution (dans laquelle les segments sont manipulés par les applications) et la mémoire permanente. Cette distinction permet de toujours maintenir, même pendant l'exécution, une version cohérente des segments en mémoire permanente.

Deux mécanismes sont fournis pour la gestion de la permanence. Le premier permet de rendre permanente l'image d'un segment. L'atomicité est offerte par le deuxième mécanisme, en l'occurrence la journalisation. Le système permet de stocker des informations dans des journaux gérés sur disque. L'utilisateur peut conserver une liste d'enregistrements dans un journal, puis après validation du journal utiliser ces enregistrements pour modifier l'image permanente de la mémoire. En cas de panne, le journal permet de reconstruire une image cohérente des modifications qui y ont été validées. Les enregistrements non validés dans le journal sont perdus. Les applications contrôlent le format et la sémantique des enregistrements écrits dans le journal, ainsi que l'opération d'exploitation du journal lors de la reprise d'un site après une panne. Une application peut ainsi gérer un journal "avant" (sauvegardant les anciennes valeurs validées) ou "après" (mémorisant les modifications validées). L'application contrôle également la mise à jour des images permanentes des segments qui peut être effectuée de manière asynchrone implicitement ou synchrone explicitement.

4 Service de protection

La mémoire virtuelle distribuée est partagée par l'ensemble des processus Unix utilisant le service de données partagées. À un instant donné, un processus Unix utilisant le service de protection s'exécute dans un domaine de protection qui définit son contexte courant d'adressage de la mémoire virtuelle partagée répartie. Deux domaines de protection différents peuvent partager des segments avec les mêmes droits ou des droits différents. Un domaine de protection est représenté par une liste de capacités, couples nom de segment - droit d'accès. Cette représentation est cachée aux applications. Lors de la première tentative d'accès à un segment dans un domaine, le système vérifie si le domaine possède une capacité sur le segment et, si c'est le cas, autorise le couplage du segment avec les droits spécifiés par la capacité.

Un domaine peut exporter vers les autres domaines des capacités d'un type spécial, appelées capacités de changement de domaine, permettant d'appeler un sous ensemble des procédures qu'il contient. Les procédures exportées sont appelées points d'entrée d'un domaine. La tentative d'appel dans un domaine *D* d'un des points d'entrée du domaine *D'* provoque le changement de domaine du processus qui a effectué l'appel si *D* contient une capacité de changement de domaine autorisant l'opération. Un changement de domaine s'accompagne en général d'un transfert de capacité du domaine appelant vers le domaine appelé pour les paramètres d'appel et du domaine appelé vers le domaine appelant pour les résultats. Ces transferts de capacités sont spécifiés par un langage d'interface (IDL).

Nous obtenons ainsi une expression de la protection indépendamment du code des applications.

5 Comparaison avec l'existant

Dans cette section, nous situons nos travaux par rapport aux projets du même domaine. Pour cette comparaison, nous passons en revue quatres axes de travail.

Gestion d'un espace virtuel unique

Plusieurs projets de recherche explorent actuellement l'utilisation des processeurs 64 bits pour la gestion d'un espace virtuel unique dans un système réparti. Il s'agit notamment des projets Opal [3], Mungi [6] et Angel [7]. Si l'utilisation des adresses virtuelles comme nom unique est un acquis, peu de choses ont été proposées pour l'allocation des adresses, la localisation ou la migration des segments dans cet espace. C'est un des aspects sur lesquels nous comptons apporter de nouvelles solutions. De plus ces projets fournissent tous des solutions incompatibles avec Unix car les processus ne peuvent pas adresser directement leur données privées. Nos services sont développés comme une extension au système AIX sans modification du système.

Cohérence et synchronisation

Le couplage fort entre la cohérence et la synchronisation est inspiré du travail réalisé dans le projet Midway [1], dans lequel la cohérence à l'entrée (*Entry Consistency*) est proposée.

La gestion de différents protocoles de maintien de la cohérence a déjà été étudiée dans des projets de recherche comme Munin [2], mais aucun système à l'heure actuelle ne permet aux applications de spécifier leur propre protocole de gestion de la cohérence. Munin laisse seulement le choix entre un nombre limité de protocoles de cohérence implantés dans le système.

Permanence

De même que pour la cohérence, les systèmes actuels ne permettent pas aux applications de spécialiser la gestion des journaux en fonction de besoins très particuliers. Nous

fournissons aux applications la possibilité d'assurer la permanence de leurs segments en mettant en œuvre une politique optimale de journalisation.

Protection

Tous les systèmes fondés sur la gestion d'un espace virtuel unique fournissent la protection sous forme de capacités logicielles et de domaines de protection (Opal [3], Mungi [6]). Toutefois, ces systèmes imposent aux utilisateurs la manipulation directe des capacités et utilisent des capacités encryptées. Ainsi, le programmeur d'application a la charge d'une partie de la gestion des capacités. Nous proposons de gérer les capacités de façon transparente à l'utilisateur, ce qui rend le code des applications indépendant de la protection.

6 Conclusion

Cet article a présenté les grandes lignes du service de gestion de données persistantes partagées qui est en cours de réalisation. Ce service permet une mise en œuvre du partage avec des protocoles de cohérence spécifiés par les applications. Il permet également de rendre ces données résistantes aux pannes en gérant des journaux avec des protocoles de journalisations spécifiés par les applications. Enfin, ces données peuvent être protégées à l'aide de capacités logicielles, la programmation de la protection étant séparée de la programmation de l'application. Nous débutons une phase d'évaluation par portage de serveur d'information, en particulier avec des gestionnaires de bases de données.

Bibliographie

[1] B. Bershad, M. Zekauskas, “Midway: Shared memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors”, *Rapport Technique CMU-CS-91-170*,Septembre 1991.

[2] J. Carter, J. Bennett, W. Zwaenepoel, “Implementation and performance of Munin”, *13th ACM Symposium on Operating Systems Principles (SOSP’91)*,pp. 152-164, Octobre 1991.

[3] J. Chase, H. Levy, E. Lazowska, M. Baker-Harvey, “Lightweight Shared Objects in a 64-Bit Operating System”, *7th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, 27(10), Octobre 1992.

[4] P. Dechamboux, D. Hagimont, J. Mossière, X. Rousset de Pina, *Arias : un service de gestion de données persistantes partagées*, (RT-2-95), Laboratoire IMAG-LSR - projet Sirac, Octobre 1995.

[5] M. J. Feeley, J. S. Chase, V. R. Narasayya et H. M. Levy, , “Integrating coherency and recoverability in distributed systems”, *Proc. of the 1st Symposium on Operating Systems Design and Implementation*,pp. 215-227, 1994.

[6] G. Heiser, K. Elphinstone, S. Russell, J. Vochteloo, *Mungi: a distributed single address-space operating system*, (TR 9314), School of Computer Science and Engineering, University of New South Wales, Novembre 1993.

[7] K. Murray, T. Stiemerling, T. Wilkinson, P. Kelly, *Angel: resource unification in a 64-bit micro-kernel*, (TR), Department of Computing, Imperial College London, Juin 1993.