Un service de gestion de données persistantes partagées adaptable

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

INRIA Rhône-Alpes - Projet SIRAC

655 venue de l'Europe - 38330 Montbonnot Saint-Martin

email: Daniel.Hagimont@imag.fr

1 Introduction

De nombreux projets de recherche travaillent à la réalisation d'un multiprocesseur à mémoire virtuelle commune sur un réseau local de machines. Cette mémoire commune peut être utilisée pour stocker des données partagées et persistantes, comme support de serveur d'information et d'applications coopératives.

Des systèmes de ce type existent déjà. Le principal problème qui se pose aujourd'hui est l'efficacité de ce support : comment exploiter au mieux les ressources disponibles dans un réseau local de machines.

Dans ce but, le service mémoire que nous fournissons est adaptable. Il fournit une architecture dans laquelle des protocoles de gestion mémoire spécifiés par l'utilisateur peuvent être intégrés dans le système. Ces protocoles déterminent la gestion de la mémoire qui convient le plus à l'application considérée.

Le contrôle de la gestion mémoire peut se faire à deux niveaux :

- La gestion des caches. Elle consiste à déterminer où, quand et comment l'information est conservée dans les caches (la mémoire physique) des machines du réseau.
- La gestion des données permanentes. Elle détermine la manière dont sont gérées les informations permanentes sur disques, ainsi que la politique de chargement et mise à jour de ces données.

Pour simplifier la réalisation des protocoles, nous fournissons aux applications un ensemble de services qui implantent les fonctions de base de la gestion mémoire. Le programmeur d'application peut contruire ses protocoles à partir de ces fonctions de base, puis intégrer ces protocoles dans le système.

Nous présentons dans les sections suivantes les principes de base de ce système (section 2), puis les services fournis respectivement pour la gestion des caches et la gestion de la permanence (section 3 et 4). La section 5 présente les expérimentations réalisées et nous concluons en section 6.

2 Principes de base

L'abstraction principale de notre système est une mémoire virtuelle partagée répartie offrant l'illusion d'un espace mémoire unique. Les allocations de mémoire partagée se font dans un bloc de 2^{63} octets (l'objectif est d'utiliser des machines 64 bits) 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 contiguës. 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.

Les données gérées dans cette mémoire peuvent être rendues permanentes sur disque, afin qu'elles survivent à un arrêt de machine ou à une panne. De même que pour la gestion en mémoire, le système fournit des mécanismes permettant aux utilisateurs de spécifier le protocole de gestion de la mémoire permanente qu'ils souhaitent.

3 Gestion des caches

La gestion des caches consiste à gérer les données partagées au niveau de mémoire auquel elles sont adressées par les processeurs. Les mécanismes fournis doivent permettre de résoudre les problèmes suivants :

- mise en cohérence des copies. Le partage des données entre des processus sur des machines différentes étant réalisé par copie dans les mémoires des machines, il est nécessaire d'assurer la cohérence entre ces copies.
- efficacité du cache. Pour être utilisée, une donnée doit être présente en mémoire. Comme toutes les données ne peuvent tenir en mémoire, le système de pagination requisitionne les pages qui ne sont pas utilisées souvent lorsqu'il a besoin de place. Cette gestion de la pagination doit faire en sorte que la plupart des objets accédés se trouvent dans le cache, minimisant les défauts de pages en mémoire.

3.1 Mise en 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.

3.2 Efficacité du cache

Un cache est efficace lorsqu'il contient l'information la plus souvent accédée. Lorsque le système est amené à faire de la place dans le cache, la page requisitionnée est généralement sauvée dans une zone temporaire sur disque appelée *Swap*. Toutefois, il est souvent possible d'adapter cette sauvegarde au comportement de l'application.

Dans notre système, nous permettons aux applications d'associer un traitant aux événements de pagination pour les segments qu'elles gèrent. Ce programme peut choisir de sauver la page dans le *Swap*, sur l'image permanente du segment concerné, ou encore de copier les zones modifiées de la page dans un journal de modifications qui est consommé de façon asynchrone. L'application peut également recopier les zones modifiées dans le cache d'une autre machine, évitant ainsi une entrée-sortie disque.

4 Gestion de la permanence

La gestion de la permanence consiste à gérer les données sur des disques, afin de tolérer un arrêt de machine ou une panne.

Les mécanismes fournis doivent résoudre les problèmes suivants :

- stockage des données sur disque. Il est nécessaire de permettre le stockage de données sur les différents disques disponibles sur les machines du réseau.

- atomicité des mises à jour. Le système doit permettre une mise à jour atomique des données sur disque, afin de parer à une éventuelle panne lors de la recopie de données.

Pour gérer des segments sur disque, notre système gère des volumes sur les disques des machines. Un volume est local à un site et contient des segment permanents. Des fonctions permettent de créer, lire et écrire des segments dans les volumes. Ces fonctions sont utilisées dans les protocoles de gestion mémoire pour gérer l'image permanentes des segments. Le protocole de gestion mémoire défini par l'application est libre de charger les données par segment, par page ou de faire de la lecture anticipée. Il gère également la répercussion des modifications sur les images permanentes des segments.

Pour assurer les mises à jour atomiques des segments permanents, 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).

5 Expérimentations

Le système décrit ci-dessus a été réalisé par adaptation du système Unix AIX sur des machines Bull de type Escala/Estrella à base de processeurs PowerPC 60x. Les protocoles de gestion mémoire sont installés dans le système AIX sous la forme d'extensions noyau (kernel extensions) et modules *streams*.

Afin d'évaluer ce prototype, nous avons réalisé une application qui consiste en un système de gestion de fichier réparti s'exécutant sur une architecture de type *cluster*. Un serveur de fichier est accessible à des machines clientes par des montages NFS. Ce serveur de fichier s'exécute sur une grappe de machines interconnectées par un réseau local. L'intérêt de cette architecture réside dans la possibilité de faire évoluer progressivement la composition matérielle du serveur de fichier (montée en puissance ou remplacement), par opposition au remplacement d'un serveur centralisé. Ce serveur de fichier appelé CFS pour *Cluster File System* donne la visibilité d'un système de fichier unique. Les machines clientes dialoguent avec des machines différentes du cluster, ce qui répartit la charge entre les machines de la grappe.

Les fichiers sont représentés par des segments qui sont partagés entre les machines. Les segments sont permanents et répartis entre les volumes. Une première version est opérationelle dans laquelle un segment est géré dans un volume unique et partagé entre les machines avec un protocole de gestion de la cohérence du type lecteurs-rédacteur.

Afin de démontrer l'adaptabilité de notre service système, deux autres protocoles de gestion mémoire sont en cours de réalisation :

- un protocole adapté aux besoins des fichiers à accès séquentiels. Dans ce protocole, un fichier est découpé en blocs qui sont gérés dans des volumes sur des disques différents. Ainsi, lorsqu'un fichier est lu séquentiellement à froid (sans être dans le cache), les lectures séquentielles provoquent des lectures en parallèles sur des disques différents, ce qui répartit la charge du chargement du fichier sur ces disques.
- un protocole optimisant la gestion du *Swap*. Lorsqu'une page de la mémoire est réquisitionnée, elle est copiée dans le cache d'une autre machine du cluster, ce qui remplace une entrée-sortie disque par une entrée-sortie sur le réseau. Ce protocole peut être très intéressant avec un réseau rapide.

Les premières mesures optenues avec ce prototype ont montré des gains d'efficacité significatifs par rapport à une solution basée sur NFS. Une expérimentation plus complète à l'aide des protocoles spécifiques mentionnés ci-dessus est en cours.

5 Conclusion et perspectives

Cet article a présenté les grandes lignes du service de gestion de données persistantes partagées adaptable que nous avons réalisé. Ce service permet une mise en œuvre du partage d'informations permanentes avec des protocoles de gestion mémoire spécifiés par les applications.

Si nous poursuivons les expérimentations avec le premier prototype, ses lacunes ont déjà été identifiées et une seconde version est en cours de réalisation. Nous allons également valider notre système avec le gestionnaire de base de données orienté-objet O2 dont la machine à objet est en cours de réalisation sur notre système.