Migration dynamique de processus et de données pour la gestion globale des ressources d'un NoW

F. André, C. Morin et M.T. Segarra

IRISA
Campus de Beaulieu
35042 Rennes Cedex
Tél : +33 (2) 99 84 00 00
Fax : +33 (2) 99 84 71 71
E-mail : {fandre,cmorin,segarra}@irisa.fr.

RÉSUMÉ. *En général dans un NoW chaque utilisateur travaille sur sa station indépendamment des autres utilisateurs. Il est alors possible qu'un utilisateur ait besoin de plus de ressources que celles offertes par sa station tandis que des ressources sont sous-utilisées dans d'autres stations du réseau. Ce document montre que l'intégration de la gestion de la ressource mémoire et de la gestion de la ressource processeur permet de décider la migration de processus et/ou des données en fonction de la charge du processeur, de l'occupation mémoire et du patron d'accès aux données.*

ABSTRACT. *Traditionally in a NoW each user works on his workstation independently of other users. So, it is possible that a user needs more resources than those offered by his workstation, whereas the resources of other workstations may be under-used. This paper shows that integrated memory and processor management allows to define a policy to choose process or data migration according to machine load, memory occupation, and process data access pattern.*

MOTS-CLÉS : *NoW, gestion de ressources, pagination sur mémoire distante, migration de processus.*

KEYWORDS: *NoW, resource management, remote memory paging, process migration.*

1. Introduction

Dans un réseau de stations de travail ou NoW chaque utilisateur travaille sur sa machine indépendamment des autres utilisateurs. Il peut alors arriver que l'un d'eux ait besoin de plus de ressources que celles fournies par sa station alors que des ressources sont sous-utilisées dans d'autres stations du réseau. Il est donc intéressant de fournir des mécanismes qui permettent d'exploiter globalement toutes les ressources d'un NoW.

Différents mécanismes permettant le transfert de l'exécution d'un processus d'une station à une autre ont déjà été proposés ([BBGL 96] et [Cab 96]) ; quelques implantations de partage de la ressource mémoire existent également (entre autres [MD 96]). Notre système présente l'originalité d'intégrer ces deux types de mécanismes ce qui permet une grand souplesse d'utilisation des ressources et l'obtention d'un coût de migration réduit.

Nous présentons dans la section 2 la politique générale que nous avons conçue et la section 3 détaille quelques aspects de la mise en œuvre. La conclusion précise les conditions d'expérimentation.

2. La ressource processeur et mémoire dans un NoW

Dans notre système, un processus regroupe deux composantes : le *contexte d'exécution* (pile d'exécution et registres du processeur), et le *contexte permanent* (code et données du processus). Un tel processus peut *a priori* s'exécuter sur n'importe quelle station et son contexte permanent peut être réparti entre plusieurs stations du NoW.

Lorsque la décision de migrer un processus est prise, son contexte d'exécution est transféré vers la station destinataire. Cependant le contexte permanent n'est pas déplacé de manière immédiate (*cf.* 3.2.).

Même si *a priori* le contexte permanent d'un processus peut être dispersé partout dans la mémoire physique du réseau, il est souhaitable de limiter cette dispersion afin de minimiser le nombre d'accès au réseau et donc le temps d'exécution. Pour mettre en place cette politique nous implantons dans chaque station du réseau deux serveurs : le *serveur de mémoire* qui gère l'utilisation de la mémoire physique du réseau et le *serveur de contexte* qui gère les contextes d'exécution des processus. Ces serveurs coopèrent entre eux sur un même site et avec leurs homologues entre sites distants.

3. Les principales fonctions du système réalisé

3.1. *Lancement de l'exécution d'un processus*

Lorsqu'un utilisateur lance l'exécution d'un processus, le serveur de contexte estime quelles sont les stations qui peuvent héberger un nouveau contexte d'exécution. Simultanément, le serveur de mémoire calcule l'ensemble de stations susceptibles de contenir entièrement le contexte permanent du processus. Si l'intersection de ces deux ensembles n'est pas vide, alors le processus peut être entièrement contenu dans une seule station et son exécution est lancée sur celle-ci.

Si l'intersection des deux ensembles est vide, il n'est pas possible d'héberger tout le processus sur la même station. Dans ce cas, l'exécution du processus est lancée sur la station qui offre la plus grande quantité de mémoire parmi celles qui sont capables de l'exécuter. Le reste du contexte permanent est réparti entre les stations qui offrent le plus de mémoire disponible de façon à minimiser la dispersion. Cet algorithme permet de regrouper au maximum le contexte d'exécution et le contexte permanent.

3.2. *Gestion de la mémoire*

Lorsque le processus est placé sur le réseau, il réalise des références à la mémoire. Si les données référencées se trouvent dans la mémoire physique locale, elles sont accédées immédiatement. Dans le cas contraire, un défaut de page est reçu par le serveur de mémoire local qui connaît l'identité de la station contenant la page fautive à qui il envoie une requête demandant cette page. Lors de son arrivée, la page est stockée dans la mémoire physique, s'il en reste de disponible. Le contexte permanent existant dans la mémoire physique de la station augmente d'une page ce qui permet de diminuer le nombre de défauts si le processus a une bonne localité de référence.

S'il n'existe pas assez de mémoire physique disponible sur la station, deux situations sont possibles :

• Une partie du contexte permanent est contenue dans la mémoire physique de la station. Ainsi, une des pages de ce contexte est échangée avec la nouvelle page. La page à remplacer est choisie en utilisant une politique de remplacement de pages comme celle utilisée dans les échanges entre mémoire et disques magnétiques.

• Il n'y a pas de contexte permanent dans la mémoire physique de la station. Il faut libérer une page du contexte permanent d'un autre processus. Le serveur de mémoire local essaie en priorité de faire migrer une page du contexte permanent d'un processus qui n'est pas exécuté par cette station (en faisant migrer cette page vers la station d'exécution du processus). Différentes autres alternatives sont possibles.

3.3. *Concept d'attraction*

La politique globale de gestion de ressources présentée précédemment essaie de rapprocher les deux composantes d'un processus. Néanmoins, il existe des cas extrêmes où le regroupement entier n'est pas possible. Afin d'améliorer ces situations, les pages distantes du contexte permanent d'un processus sont amenées vers sa station d'exécution ce qui permet d'exploiter la localité de référence des accès. Cependant, dans le cas où les références mémoire suivent un patron régulier il peut être préférable de faire migrer le contexte d'exécution vers la station qui contient le contexte permanent qui est en train d'être accédé. C'est ce que nous appelons *l'attraction*.

4. Expérimentation

Des tests ont été réalisés sur une plate-forme composée de quatre stations PC Pentium sous Unix reliées par un réseau Ethernet à 10 Mbps. Entre 10 et 20 applications ont été lancées en parallèle et ont subi des migrations de données et/ou de contexte en cours d'exécution. Le temps moyen de résolution d'un défaut de page est de 9.4 msec pour une taille de pages de 4Ko. Ce temps est similaire au temps de résolution d'un défaut de page sur un disque magnétique et pourrait être largement amélioré avec un réseau plus performant. Le compromis migration de contexte/pagination à distance est montré sur la figure 1 pour le cas de matrices de 512x512 éléments. En abscisse figure le pourcentage de données à distance. Pour chacun de ces cas, la barre à gauche correspond au cas où l'application effectue de la pagination à distante et la barre à droite représente le cas où l'application est migrée sur le site contenant les données distantes. On constate le coût dû à la pagination (en pourcentage du temps d'exécution).

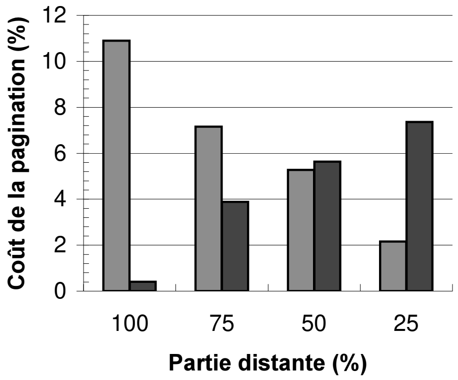

FIG. 1 – *Pagination vs. migration pour Matmul 512x512*

5. Conclusion

Une première mise en œuvre des deux mécanismes présentés dans ce document a été effectuée. Cette version souffre de quelques limitations, notamment c'est au programmeur de spécifier les points où la migration est possible. Ce travail va être repris pour gérer des processus dans le cadre d'un réseau mobile sans fil.

Références

[BBGL 96] BARAK A., BRAVERMAN A., GILDERMAN I., LA'ADAN O., *The MOSIX Multicomputer Operating System for Scalable NOW and its Dynamic Resource Sharing Algorithms*. Rapport technique 96-11, Institute of Computer Science, The Hebrew University, juillet 1996.

[Cab 96] CABILLIC G., *Exécution d'applications parallèles sur architectures distribuées hétérogènes : propositions et mise en œuvre*. Thèse de doctorat, Université de Rennes 1, octobre 1996.

[MD 96] MARKATOS E.P., DRAMITINOS G., *Implementation of a Reliable Remote Memory Pager*. *In : Proc. of the USENIX 96 Technical Conference*. San Diego, Ca, janvier 1996.