Etude et résalisation d'un intergiciel pour la résilience d'applications parallèles distribuées sur un intranet
Rapport final
Mai 2004

Jean Michel Nlong et Yves Denneulin
ID-IMAG, ENSIMAG, Montbonnot

11 mai 2004

1 Introduction

Traditionnellement le calcul haute performance s'appuyait sur des solutions coûteuses, basées sur des processeurs et des logiciels spécifiques, et destinées à un nombre limité de domaines tels que la météorologie, les phénomènes physiques, le nucléaire...Cela exigeait que les centres de recherche disposassent d'une puissance de calcul sans cesse croissante, qui n'était pas accessible à des organisations moyennes.

L'évolution de la technologie des microprocesseurs permet aujourd'hui à des ordinateurs de bureau standard d'atteindre un tel niveau de performance qu'il est possible de construire à partir de ces machines des supercalculateurs aussi rapides que les calculateurs dédiés de type CRAY, IBM SP ou SGI Origin 2000. Avec l'avèment du logiciel libre, et le coût assez bas des ordinateurs de bureau, la très forte demande en calcul haute performance (scientifique et technique) peut être satisfaite. Les grappes de PC apparaissent ainsi comme une très bonne alternative aux autres moyens de calcul.

Une grappe (de PC) est un ensemble de PC reliés en réseau, considéré comme une ressource unifiée de calcul. Le procédé consiste donc à déployer des PC ordinaires, (jusqu'à plusieurs centaines de machines) connectés à travers un réseau haut débit (Fast Ethernet par exemple), et à fédérer le fonctionnement de ces machines par des logiciels d'exploitation globale (notamment ordonnancement, équilibrage de charge, attribution et partage de ressources).

Toutes les entreprises disposent aujourd'hui de leur propre réseau local privé, auquel sont connectées toutes les machines du groupe. Ces machines sont affectées à différents services administratifs, production etc., et sont

très souvent inexploitées pendant de longs moments, tels que le sfins de semaine, les congés, les nuits et les absences du personnel. Le grand nombre de machines inexploitées représente une puissance de calucul considérable. L'exploitation de ces périodes de jachère pour des tâches de production permettrait d'économiser un investissement sur du matériel nouveau, dédié. Toutefois le caractère volatile des ressources ainsi collectées pose de nombreux problèmes...

La disponibilité de ces ressources est extrêmement dynamique, ce qui pose le problème de tolérance aux fautes du fait qu'un programme qui démarre sur une station n'est pas assuré de pouvoir terminer, la machine pouvant disparaître à tout moment. L'exploitation effective de ces ressources nécessite de disposer d'applications s'adaptant aux départs et aux arrivées intempestifs de ressources, sur des architectures très variées (matériel, système, réseau).

Le projet I-Cluster a pour objectif la détection et l'exploitation de jachères de ressources au sein d'un intranet pour du calcul intensif. Bruno Richard[1] a étudié la faisabilité d'un tel système permettant de fédérer de façon transparente pour les utilisateurs les ressources d'un réseau d'entreprise, afin de les agréger sous forme de grappe virtuelle. Nous nous intéressons pour notre part à offrir sur ce système un service de déploiement d'applications fiable et robuste, pour des applications distribuées, c'est-à-dire s'exécutant sur plusieurs sites communicants à travers le réseau sous-jacent.

Ce travail porte sur la conception d'un intergiciel permettant le déploiement d'application tolérant les déconnexions de ressources par des mécanismes de sauvegarde/reprise ou de réexécution partielle, ou encore de migration de processus. La migration des processus d'un nœud en panne vers un autre tente de trouver une solution à ces problèmes ; un processus peut ainsi survivre à un plantage du processeur sur lequel il s'exécute : si une image cohérente de son contexte a été transférée sur un autre processeur avant la panne, il pourra être repris sur l'autre machine.

2 Contexte scientifique

Plusieurs travaux se sont intéressés à la migration de processus, dans des contextes différents. Pour une discussion détaillée voir[2]. Malgré la demande importante, la migration de processus reste encore assez peu répandue. La plupart des systèmes qui se sont intéressés à la migration de processus se limitent à une classe de problèmes et de plate-formes précises et ne sont pas réutilisables dans d'autres environnements.

2.1 schéma général

Milojicic[2] résume les schémas classiques de migration de processus en huit phases[1] :

1. Une requête de migration est déclenchée sur un nœud distant. Après négociation, la migration est acceptée,
2. l'exécution du processus concerné est suspendue sur le nœud origine. Il est marqué comme "en migration",
3. les communications éventuelles sont prises en compte pendant tout le reste de la procédure par un autre processus (les messages sont tout juste sauvegardés, pour être rejoués après le reprise du processus),
4. le contexte actuel du processus est extrait ; cela inclut le contexte du processeur, le contexte mémoire, les fichiers ouverts et les canaux de communication,
5. une instance est créée sur la machine destination, destinée à recevoir le contexte du processus transféré,
6. le contexte est importé sur la nouvelle instance,
7. une forme de suivi des références peut être laissée sur la machine source (pour la communication ou le contrôle du processus),
8. la nouvelle instance est reprise lorsque l'on dispose d'assez d'éléments pour son exécution. Lorsque tout le contexte est transféré, le processus peut être détruit du nœud origine.

Ces services peuvent être implémentés à plusieurs niveaux d'une architecture (comme partie intégrante de l'application, comme bibliothèque utilisateur, ou au niveau du système d'exploitation), avec des performances différentes au niveau de la transparence aux applications, la portabilité sur plusieurs systèmes d'exploitation, la généricité ou la prise en compte de plusieurs types d'applications :

- Une façon naturelle de permettre la migration d'une application est de la prendre en compte dans le codage de l'application. [4] propose un schéma d'auto-sauvegarde du contexte à des instants précis de son exécution tels que les appels de fonctions, les créations d'objets...Cette forme de migration est limitée à l'application elle-même ; toutefois l'implémentation peut être simplifiée, et optimisée pour certains cas
- La migration en mode utilisateur permet de pallier l'absence de support au niveau système pour la migration de processus. Elle est destinée à des applications sollicitant peu de ressources système. L'exemple caractéristique de ce courant est Condor[5]. Condor implémente les mécanismes de migration dans une bibliothèque en mode utilisateur.

[1] Ce schéma est très général, et nous verons par la suite qu'il ne peut être rigoureusement appliqué à notre problème

Condor nécessite que le programme à surveiller soit lié avec la bibliothèque de migration. À la réception d'un avis de checkpoint, l'application crée un processus fils qui enregistre son contexte pendant que le processus original (père) continue son exécution. La bibliothèque de checkpoint est réalisée en modifiant (augmentant) certains appels système pour enregistrer les valeurs de retour et maintenir une trace de tous les objets utilisés par l'application. Lorsqu'un processus quitte son environnement d'origine, il y laisse un résidu à qui il fait suivre les requêtes destinées au système.

- La prise en compte de la migration de processus au niveau système requiert de profondes modifications du noyau sous-jacent. Le système le plus connu dans cette catégorie est Mosix[6], un système distribué conçu au Hebrew University of Jerusalem. Le partage dynamique de charge est réalisé dans Mosix par la migration de processus. Le noyau Mosix[6] consiste en deux parties : un mécanisme de migration préemptive de processus (Preemtive Process Migration PPM) et des mécanismes de partage de ressource. Au moment de la commutation de contexte, le PPM peut choisir de migrer n'importe quel processus sur un tout nœud disponible. Chaque processus possède un nœud domicile unique (Unique Home Node UHN), qui est celui où il a été créé. Les processus ayant migré sur un autre nœud utilisent les ressources locales autant que possible, mais interagissent avec le système uniquement à travers leur UHN. Ainsi, comme dans Condor, les appels système émis par le processus seront routés vers le UHN.
- Crak[8] est un système qui vise à combiner les avantages de Mosix et Epckpt[7]. Dérivé de epckpt, il implémente la migration de processus sous la forme d'un module, donc ne nécessite pas de modification du noyau. Les processus sont autorisés à utiliser toutes sortes de ressources (tubes, signaux, fichiers...).

2.2 Contraintes de l'environnement I-Cluster

I-Cluster est constitué de ressources extrêmement dynamiques, les machines qui constituent la plate-forme adhèrent et quittent le nuage de façon intempestive, et il n'est pas facile de diagnostiquer les pannes. Toutefois le nombre de machine est potentiellement grand, et il est toujours possible de trouver une station de travail où faire exécuter un calcul. Il n'y a néanmoins pas de garantie que ce calcul se termine normalement.

Une solution de type Mosix ne peut cependant être applicable dans notre cas :

Tout d'abord lorsqu'une machine disparaît du nuage, elle disparaît complètement. La notion de Unique Home Node n'a donc pas de sens.

De plus pour des raisons de portabilité, nous avons choisi d'ajouter des fonctionnalités à un noyau existant, plutôt que de le modifier.

Une solution de type Condor n'est pas non plus envisageable. I-cluster est une plate-forme ouverte, et nous n'avons pas de contrôle sur les applications clientes. On ne saurait donc ni les recompiler, ni même en inspecter le code.

La qualité souvent approximative des ressources matérielles disponibles (processeurs, réseaux, périphériques) doit être compensée par les possibilités d'allocation dynamique des processeurs aux processus. L'objectif de pérennité du service est souvent beaucoup plus fort que toute autre considération. Trois points spécifiques ont guidé notre travail : assurer la transparence de la migration, supporter l'hétérogénéité des architectures et privilégier le support efficace d'applications parallèles. Aucun des systèmes connus à ce jour ne permet d'atteindre ces objectifs à la fois, en particulier la prise en compte des communications à la migration, dans un contexte de disparition complète du nœud origine de l'application.

Nous avons opté pour une solution de sauvegarde/reprise du contexte d'un processus au niveau du noyau Linux en étant le moins intrusif possible au niveau de la configuration, c'est-à-dire sans nécessiter de modification du noyau sous-jacent ni de recompilation/édition du programme à surveiller. Les capacités des modules Linux offrent une alternative intéressante, puisqu'il est ainsi possible d'ajouter de nouvelles fonctions au noyau sans le recompiler. Ces capacités étant implémentées au niveau du noyau, on peut surveiller et migrer n'importe quel type d'application.

3 Réalisation

Le système (samory)[2] est conçu en trois couches logicielles : des modules pour noyau Linux qui permettent d'observer et de récupérer le contexte des processus pris en charge ; une bibliothèque d'appels systèmes pour l'interfaçage avec les modules et différents utilitaires pour contrôler le système.

Le schéma de migration que nous avons suivi est le suivant ; il diffère un peu de celui vu en 2.1 :

1. Négociation de la migration sur un nœud distant : Cette phase est essentielle. Etant donné que dans notre cas un processus est migré lorsqu'intervient un crash sur le nœud origine, une requête de migration est déclenchée sur un nœud distant. La négociation est donc conduite et acceptée bien avant l'avis de migration,
2. à intervalles réguliers, le processus surveillé est suspendu sur le nœud origine, son contexte est extrait et sauvegardé sur le nœud destination, en préparation à la migration,
3. pendant l'exécution des processus, les communications sont enregistrées. Les messages non encore lus par l'application sont aussi sauvegardés sur le nœud destination, pour être rejoués après le reprise du

[2]Le système était d'abord baptisé Miyunga.

processus,

4. après un plantage de la machine source, une instance est créée sur la machine destination, et son contexte est remplacé par le contexte du processus transféré,
5. les autres processus communicants avec le "migrant reçoivent notification de la migration, et mettent à jour les canaux de communications impliqués,
6. les messages reçus entre la dernière sauvegarde et la reprise sur un nouveau nœud sont rejoués au "nouveau processus." Aucune forme de suivi des références ne peut être laissée sur la machine source, puisque celle-ci n'est plus joignable.

3.1 Support du noyau Linux

Le module noyau réalise la migration des ressources suivantes :

l'espace d'adressage : la cartographie mémoire doit sembler inchangée à la reprise du processus. Le "nouveau" processus doit pouvoir générer et accéder aux mêmes adresses avec les mêmes attributs de protection. À la reprise du processus sauvegardé, nous disposons d'un processus enveloppe qui nous "prête" son contexte. Nous restaurons l'espace d'adressage par remplacement de ce contexte par notre sauvegarde (segments de code, données initialisées, données non initialisées, arguments, environnement, pile).

les signaux : encore appelés interruptions logicielles, ce sont des mécanismes qui permettent à un processus de se synchroniser avec d'autres processus ou événements du système, et de prendre une action appropriée (Terminaison d'un processus fils, expiration d'une temporisation, etc.). Nous gérons tous les types de signaux supportés par les applications Linux.

le contexte du processeur : il est constitué de l'ensemble des valeurs des registres du processeur relativement à l'application en cours d'exécution. Ces registres contiennent entre autre l'instruction en cours d'exécution et ses paramètres, ainsi que le prochaine instruction. La reprise de l'exécution doit commencer avec ces valeurs, qu'il faut par conséquent réintroduire dans les registres du nouveau processeur. Les deux processeurs (origine et cible) doivent donc être compatibles. Le système samory a été conçu pour la famille de processeurs Intel x86 et compatibles. La reprise des appels systèmes est un cas particulier, puisque leur code ne fait pas partie du contexte mémoire de l'application, et que les résultats ne sont disponibles qu'à la fin complète de l'appel. Lorsque le système se trouve en présence d'un appel système, ce dernier est tout simplement redémarré.

les fichiers réguliers : la sauvegarde et la reprise doivent permettre au processus de retrouver les mêmes descripteurs associés aux mêmes fichiers. Le système restaure les fichiers ouverts au moment de la sauvegarde, avec les mêmes permissions et drapeaux. Dans l'état actuel, il ne prend pas en charge le transfert des fichiers de la source à la destination, et suppose que le fichier est accessible par les mêmes moyens (système de fichiers de type NFS, disponibilité du fichier sur la machine locale avec le même chemin). Il ne vérifie pas non plus la cohérence des données.

les tubes : une des formes de communication inter-processus est l'utilisation de tubes. Au niveau du programmeur, un tube se présente comme un couple de fichiers, un en lecture et l'autre en écriture. Pour communiquer, un des processus écrit des données sur le tube, que l'autre peut lire. Au niveau système il s'agit de tampons alloués dans la mémoire noyau des processus. La migration de processus communicant par des tubes nécessite que les deux processus soient restaurés sur la même machine, et au même moment. Samory prend en compte cette contrainte.

3.2 Support des communications réseau

Un des aspects les plus critiques de migration de processus parallèles est le support des communications réseau ouvertes au moment où l'un des processus est déplacé. La fermeture ou la disparition brutale de l'une des extrémités de la connexion entraîne la cassure du protocole de communication et peut rendre impossible la continuité des processus impliqués. Nous nous sommes intéressés aux processus communicant à travers le protocole TCP. La difficulté à migrer de tels processus vient de ce que TCP maintient un ensemble complexe de structures de données pour chaque connexion, et que la modification des ces données impose de modifier le code noyau TCP. Nous avons choisi une autre alternative : nous utilisons Netfilter, le module de filtrage du trafic réseau de Linux pour capturer les paquets échangés entre les processus et nous avons construit des outils au dessus de ce module pour sauvegarder et restaurer ces communications. Ce système présente plusieurs avantages :

- le code réseau du noyau n'est pas modifié, ce qui garantit la stabilité et la portabilité du système,
- la gestion des paquets réseau se fait en mode utilisateur, ce qui permet de stocker de plus grandes quantités d'informations sur les messages échangés,
- la possibilité de stocker les messages permet une gestion plus souple du checkpoint/restart ; il n'est plus nécessaire de stopper tous les processus de l'ensemble au moment de prendre l'image de l'un d'eux. Pour retrouver un contexte cohérent par rapport au reste des processus,

les messages transmis entre la dernière sauvegarde et la migration du processus sont rejoués.

4 Tests et performances

Samory a été testé sur quelques programmes disponibles sur une architecture Linux. L'environnement de tests est constitué de PC usuels de générations différentes. Nous avons redémarré les processus indépendamment de la machine où a été faite la sauvegarde. Les mesures sont conduites dans deux circonstances différentes : lorsque tout l'espace d'adressage est sauvegardé, et lorsque nous ne sauvegardons pas le code et les bibliothèques partagées ; dans ce dernier cas cet espace est reconstitué à partir de l'environnement local. Nous mesurons chaque fois la taille des données, le temps de sauvegarde, et le temps de reprise. Les temps de sauvegarde et de reprise sont de l'ordre de quelques millisecondes, ce qui est intéressant pour des programmes clients du système. Cette version du système (sans prise en charge réseau) a fait l'objet d'une publication à la troisième conférence française sur les systèmes d'exploitation en octobre 2003 [9]. Un autre article est actuellement en cours de rédaction sur les détails du protocole de migration des communications. Le système à l'état actuel n'intègre pas les outils de déploiement et de gestion automatique des processus, ni de détection de défaillances ; ils sont en cours de développement. L'administrateur du système doit lui-même prendre les décisions liées à cette gestion : arrêter un processus, sauvegarder le contexte, redémarrer etc. Samory a été testé avec succès avec les noyaux Linux 2.4.2 à 2.4.22, distribution Mandrake. Nous avons obtenu des résultats satisfaisants avec une grappe quatre machines HP-Evectra connectées à travers un réseau Ethernet 100. Trois processus tournent sur les machines, et de temps en temps un des processus est migré sur la machine "libre".

5 Utilisation

5.1 Obtenir les sources

http ://www-id.imag.fr/Laboratoire/Membres/Nlong2_ Jean-Michel/

5.2 Compiler et installer le système

Configuration système requise :

1. Linux noyau 2.4 ou plus
2. Support Filtrage de paquets Netfilter
3. Bibliothèque libipq installée (pour capturer les paquets réseau)

Compilation des sources Sur chacune des machines du cluster :

1. Extraire les sources : tar -xvf samory.tar <où-l'on-veut> installe les sources.
2. make config (inlcus certains symboles du noyau dans le projet. S'assurer que le fichier /boot/System.map existe et pointe sur le noyau actuellement en exécution)
3. make libs
4. make samory
5. make install

5.3 Exploitation

Se placer dans le répertoire <où-l'on-veut>. Avec les privillèges root, faire :

1. ./install <liste des autres machines impliquées> : installe le module, charge le module ip_queue, ajoute les règles de filtrage pour les segments TCP de ou vers <liste des autres machines impliquées>
2. ./samory init -i : initialise samory. Le système est prêt.
3. ./samory add <pid> : ajoute le processus de pid <pid> à la liste des processus surveillés par samory
4. ./samory checkpoint <file> <comfile> : Extrait le contexte de tous les processus actuellement pris en charge. Il est possible d'extraire un seul processus. mais l'option n'est pas encore ajoutée.
5. ./samory restart <file> <comfile> : Redémarre les processus contenus dans <file> sur le nœud local. <comfile> contient les communications.
6. ./samory quit : Arrête samory et libère les ressources utilisées (Les processus surveillés ne sont pas touchés).

Références

[1] Bruno Richard, Agrégation des ressources inexploitées d'un intranet et exploitation pour l'instanciation de services de calcul intensif, *Thèse de doctorat, Institut National Polytechnique de Grenoble, 12 Décembre 2003*

[2] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler and Songnian Zhou, Process Migration, *ACM Computing Surveys, Vol. 32, No 3, September 2000, 241-299*

[3] Krishna A. Bharat and Luca Cardelli, Migratory Applications, *Proceedings of the Eight Annual ACM Symposium on User Interface Software Technology, 1995*

[4] Hai Jiang, Vipin Chaudhary, Compile/Run-time Support for Thread Migration, *International Parallel and Distributed Processing Symposium, Symposium Volume April 15-19, 2002, Fort Lauderdale, California.*

[5] The Condor Project Homepage, *www.cs.wisc.edu/condor/*

[6] www.mosix.org

[7] EPCKPT, Eduardo Pinheiro Checkpoint Project, *http ://athos.rutgers.edu/ edpin/epckpt/*

[8] Hua Zhong and Jason Nieh, Crak : Linux Checkpoint/Restart As a Kernel Module *Technical Report CUCS-014601, Departemement of Computer Science, Columbia University, November 2001*

[9] Jean Michel Nlong, Yves Denneulin, Migration des processus Linux sous I-Cluster, *Proceedings RenPar'15, CFSE'3, SympAAA'2003, 15-17 octobre 2003, La Colle sur Loup, France, pp 614-623*