Répartition de Charge au dessus du Micro-noyau Mach : Intégration de la Migration dans le Système Masix

Franck Mével, Julien Simon
Franck.Mevel@masi.ibp.fr, Julien.Simon@freenix.fr

Introduction

La distribution de charge est un problème étudié depuis longtemps [Bokhari 1979]. Différentes approches de placement et de migration de processus ont été implémentées, soit à l'intérieur du système d'exploitation [Douglis et Ousterhout 1991], soit à l'extérieur [Folliot et Sens 1994]. Depuis quelques années, la conception de systèmes s'oriente vers des approches à base de micro-noyau [Accetta *et al.* 1986, Tanenbaum *et al.* 1990, Rozier 1992], ce qui facilite l'intégration de la migration à l'intérieur du système. Cependant, la collecte d'informations pour le calcul de la charge est très délicate. En effet, les systèmes basés sur un micro-noyau sont souvent composés de plusieurs serveurs qui ne possèdent chacun qu'une partie de ces informations.

Notre étude s'inscrit dans le cadre du système d'exploitation multi-serveurs Masix [Card *et al.* 1993], basé sur le micro-noyau Mach, en cours de développement au laboratoire MASI. Il est conçu pour utiliser les ressources d'un réseau de stations de travail de manière optimale. Il comporte donc des extensions réparties qui permettent aux stations de mettre leurs ressources en commun (processeurs, mémoire, périphériques, fichiers).

De plus, Masix autorise l'exécution simultanée de plusieurs environnements de travail. Il est possible d'exécuter parallèlement sur une même station des applications Unix, DOS, OS/2, Win32, etc.

Nous présentons tout d'abord la structure générale du système, puis nous étudions le mécanisme de migration qui est à la base de la répartition de charge dans Masix. Enfin, nous expliquons les problèmes liés à la collecte d'informations sur la charge dans un système multi-serveurs.

Structure de Masix

Masix est construit selon un modèle multi-serveurs, ce qui lui confère une grande modularité. En effet, chaque serveur peut être écrit et testé indépendamment des autres. De plus, il est possible d'ajouter ou de retirer dynamiquement un serveur au système.

Masix est composé de deux couches, illustrées dans la figure 1 :
- les environnements, qui implémentent les sémantiques des systèmes d'exploitation traditionnels ;
- le système d'accueil générique réparti (SAGR), qui offre des services répartis aux différents environnements (communications, mémoire répartie, gestion répartie de processus, tolérance aux fautes, ...).

Masix repose sur le micro-noyau Mach 3.0 [Tevanian et Rashid 1987]. Comme tout micro-noyau, Mach ne fournit pas toutes les fonctionnalités traditionnelles d'un système. Il offre uniquement :
- la gestion de tâches et de *threads*, qui constituent le support de l'implémentation des processus. Une tâche est une unité d'allocation de ressources, tandis qu'un *thread* est une unité d'exécution ;
- les communications entre tâches selon un modèle client-serveur, par le biais de ports et de messages. Un port est un canal de communication unidirectionnel, accessible via des droits d'émission et de réception. Un message est un ensemble de données typées transmis vers un port ;
- la gestion de la mémoire physique et virtuelle ;
- la gestion des périphériques physiques.

FIGURE 1 – Structure du système Masix

Mécanisme de migration de Masix

Peu d'études ont été menées sur la migration au dessus de Mach. [Milojicić *et al.* 1993] étudie le migration de tâches Mach. Selon les auteurs, cette approche a l'avantage d'être indépendante du système qui s'exécute au dessus du micro-noyau, puisque l'entité déplacée l'est. La migration est donc totalement transparente pour le système.

Cependant, cette technique a l'inconvénient de garder des dépendances résiduelles : chaque requête concernant la tâche (ou le processus qu'elle représente) est redirigée vers le site d'origine de la tâche. Elle repose toutefois sur une version étendue du micro-noyau incluant un nommage global et une mémoire partagée distribuée, ce qui facilite la redirection.

Cette approche est pertinente dans le cadre d'une architecture fortement couplée, comme une machine multi-processeurs, où les communications entre noeuds sont rapides et fiables, mais l'est beaucoup moins pour une architecture faiblement couplée, comme un réseau de stations de travail, où les communications sont coûteuses et sujettes à des pannes.

À partir de ces remarques, nous proposons d'étendre le mécanisme de migration en définissant dans le cadre de Masix une nouvelle entité permettant de diminuer les dépendances résiduelles : le processus générique (PG).

Chaque PG possède un nom global unique et correspond à une tâche Mach contenant un ou plusieurs *threads*. Un PG ne peut migrer que vers les sites exécutant l'environnement auquel appartient la tâche qu'il représente.

On utilise un second niveau de nommage dans les environnements, en définissant des processus locaux et des processus globaux :

- un processus local ne peut pas migrer, soit parce qu'il est lié à la machine sur laquelle il a été créé (serveur X, démons réseau), soit parce qu'il requiert de fréquentes interactions avec l'utilisateur (shells). Un processus local est créé directement par le serveur de processus de son environnement et n'est connu que du noeud sur lequel il est créé ;
- un processus global peut migrer. Il est créé par le SAGR à la demande d'un environnement et est associé à un PG. Cet événement est diffusé aux noeuds du réseau, afin qu'ils aient tous accès au processus global. Les appels le concernant ne sont donc plus redirigés vers son noeud d'origine. Ainsi, sa migration est totalement transparente et n'introduit pas de dépendances résiduelles.

Répartition de charge dans Masix

La migration est décidée par le SAGR qui utilise un algorithme multi-critères prenant en compte le temps CPU consommé, l'espace mémoire utilisé, les communications effectuées, ...

La difficulté dans notre approche consiste à recueillir les informations nécessaires pour prendre cette décision. En effet, ces données sont disséminées dans plusieurs serveurs et le micro-noyau lui-même en possède une partie, par exemple le temps CPU utilisé par un *thread*.

Chaque serveur du SAGR gère une partie de ces informations :

- le gestionnaire de processus accumule le temps CPU consommé par les PG (temps CPU des *threads* du PG + temps CPU consommé dans des appels système liés au PG) ;
- le serveur réseau comptabilise les requêtes externes de chaque PG ;
- le serveur de fichiers comptabilise les fichiers ouverts par chaque PG ainsi que ses requêtes

d'entrée/sortie.

Périodiquement, le serveur de processus générique interroge les autres serveurs pour collecter les informations relatives aux PG. Il connait ainsi les statistiques se rapportant aux processus et peut les utiliser pour décider de migrer certains PG.

Conclusion

Les architectures multi-serveurs au dessus des micro-noyaux posent de nouveaux problèmes dans le domaine de la répartition de charge.

La définition d'une couche générique répartie nous permet d'offrir une répartition de charge indépendante des différents environnements.

Notre méthode minimise les dépendances résiduelles, car un processus qui migre est pris en charge par les serveurs(SAGR et environnements) du site sur lequel il a été déplacé. Seuls quelques appels système liés aux périphériques du site d'origine sont redirigés vers celui-ci.

Références

[Accetta *et al.* 1986] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, et M. Young. Mach : A New Kernel Foundation For UNIX Development. In *Proceedings of the USENIX 1986 Summer Conference*, June 1986.

[Bokhari 1979] S. H. Bokhari. Dual Processor Scheduling with Dynamic Reassignement. *IEEE Transactions on Software Engineering*, SE(5) :326–334, July 1979.

[Card *et al.* 1993] R. Card, E. Commelin, S. Dayras, et F. Mével. The MASIX Multi-Server Operating System. In *OSF Workshop on Microkernel Technology for Distributed Systems*, June 1993.

[Douglis et Ousterhout 1991] F. Douglis et J. Ousterhout. Transparent Process Migration : Design Alternatives and the Sprite Implementation. *Software-Practice and Experience*, 21 :757–785, August 1991.

[Folliot et Sens 1994] Bertil Folliot et Pierre Sens. GatoStar : A Fault-Tolerant Load Sharing Facility for Parallel Applications. In *Proceedings of the First European Dependable Computing Conference*, pages 581–598, Berlin, October 1994.

[Milojicić *et al.* 1993] Dejan S. Milojicić, Peter Giese, et Wolfgang Zint. Experiences with Load distribution on top of the Mach Microkernel. In *Proceedings of the 4th USENIX SEDMS Symposium*, San Diego, September 1993.

[Rozier 1992] M. Rozier. Overview of the Chorus Distributed System. In *Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pages 39–70, April 1992.

[Tanenbaum *et al.* 1990] A.S. Tanenbaum, R. van Renesse, H. van Staveren, G. J. Sharp, S.J. Mullender, J. Jansen, et G. van Rossum. Experiences with the Amoeba Distributed Operating System. *Communication of the ACM*, 33 :46–63, December 1990.

[Tevanian et Rashid 1987] A. Tevanian et R. Rashid. *Mach : A Basis for Future UNIX Development*. Technical Report CMU-CS-87-139, 1987.