Simulation et évaluation de performances d'un système de répartition de charge

Yanal Hajmahmoud — Bertil Folliot — Pierre Sens

LIP6 - Équipe SRC
Université Pierre et Marie Curie
4, Place Jussieu - 75252 Paris Cedex 05
{*Yanal.Hajmahmoud,Bertil.Folliot,Pierre.Sens*}*@lip6.fr*

RÉSUMÉ. *La modélisation et l'évaluation de performance des systèmes de distribution de charge sont difficiles en raison du nombre important de paramètres à prendre en compte. Nous présentons, dans cet article, un modèle paramétrable et configurable qui simule le comportement d'un système réel de distribution de charge et prend en compte la configuration des applications, de l'environnement et des algorithmes de répartition de charge. Nous avons construit le modèle à partir d'un système réel, jusqu'à obtenir des performances par simulation quasiment identiques à celles observées en environnement réel. Ainsi, il est aisé de modifier certains paramètres de manière à extrapoler les résultats pour diverses applications et configurations matérielles. Les simulations permettent d'analyser finement l'influence des paramètres internes du système de distribution sur les performances. Les résultats par simulation sont alors re-vérifiés sur le système réel, pour le rendre adaptatif.*

ABSTRACT. *Modeling and performance evaluation of load sharing systems are difficulty due to the large amount of parameters to take into account. We present in this paper a model of a load distributing system to evaluate the performances of the system taking into consideration the applications behaviors, the network configuration, and the parameters of load balancing algorithms. The model simulates a real load balancing system. We obtain results almost similar to those obtained by the simulated system in a real environment. This work is divided into two parts. The first is the development of the model; the second is successive manipulations of the parameters and the model itself until reaching satisfied results. Once the suitable parameters have been found, which configure the simulator, it is easy to modify some of the internal parameters to extend the results of the real system. The model has also allowed to study the adaptability of load balancing algorithms and their parameters with various applications and hardware configurations.*

MOTS-CLÉS : *Équilibrage de charge, application parallèle, simulation, performance*

KEY WORDS : *Load balancing, parallel application, simulation, performances*

1. Introduction

Un des objectifs fondamentaux des systèmes distribués est de maximiser les performances en introduisant des mécanismes de distribution de charge. Ces mécanismes ont pour rôle de transférer la charge des sites surchargés vers des sites moins chargés. L'évaluation des performances d'un système de distribution de charge est un problème difficile compte tenu d'une part du nombre important de paramètres à prendre en compte (concernant le type d'applications et d'environnements répartis) et d'autre part de la difficulté d'effectuer des mesures en environnement réel. Des algorithmes et des paramètres inadaptés à un environnement d'exécution donné et à une application donnée peuvent fortement dégrader les performances, voire conduire à l'écroulement du système.

Les performances des algorithmes de distribution de charge dépendent essentiellement de trois niveaux de configuration : la configuration de la charge à exécuter, celle du système de distribution de charge et celle de l'environnement d'exécution. Dans cet article, nous proposons un modèle paramétrable et configurable qui simule le comportement d'un système réel de distribution de charge en prenant en compte ces trois niveaux de configuration.

Notre approche est de construire le modèle à partir d'un système de distribution de charge existant développé au laboratoire LIP6 (GatoStar) jusqu'à obtenir des performances par simulation quasiment identiques à celles observées en environnement réel. Le travail présenté a nécessité des raffinements successifs des paramètres des différents composants du modèle. Une fois tous les paramètres obtenus, il est aisé d'en modifier certains de manière à extrapoler les résultats pour diverses applications et configurations matérielles. Les simulations permettent ensuite d'analyser l'influence des paramètres internes du système de distribution sur les performances afin d'élaborer des nouvelles stratégies adaptatives de distribution. La figure 1 résume la méthodologie utilisée.

La section 2 décrit le système de répartition GatoStar. La section 3 présente le modèle implémenté en utilisant QNAP2. Dans la section 4, nous analysons les performances de notre modèle en analysant l'impact des nombreux paramètres. Enfin, la section 5 présente d'autres approches de simulation.

2. Le système de répartition

GatoStar est un support d'exécution d'applications parallèles un environnement multi-utilisateurs, multi-applications pour un réseau de stations de travail [FOL 95, SEN 98]. GatoStar intègre un gestionnaire de placement et de migration chargé de répartir automatiquement les applications parallèles sur un ensemble de sites. Il vise des applications à longue durée de vie et gros grain de parallélisme composées d'un ensemble de processus communiquant. Ce modèle

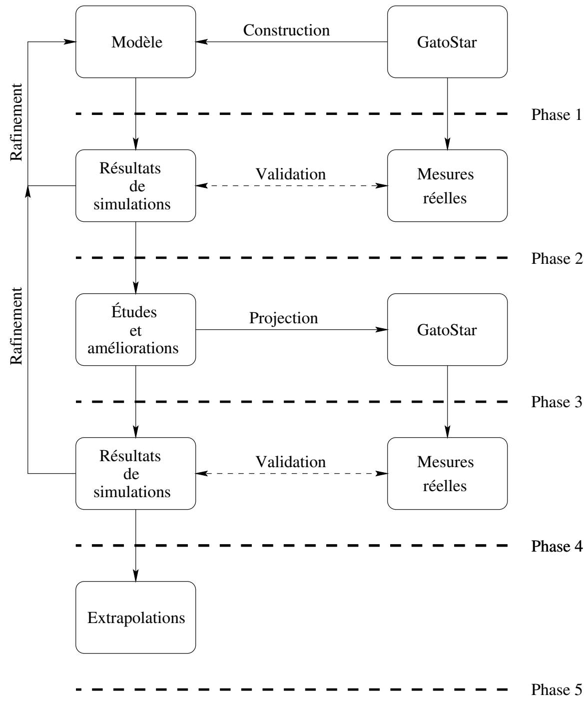

Figure 1. *Processus de développement du modèle*

couvre un large spectre d'applications scientifiques, comme la factorisation des grands nombres, la conception VLSI ou le traitement d'images.

2.1. *Classification des algorithmes de partage de charge*

La répartition de charge vise à améliorer les performances globales des systèmes répartis en allouant les nouvelles tâches sur les sites les moins chargés. La répartition peut être statique ou dynamique. Dans une stratégie statique, la décision de placement est uniquement basée sur les informations concernant le comportement moyen du système [TAN 85]. Dans les approches dynamiques, les décisions sont prises en fonction de l'état courant du système [BAR 89, ZHO 93].

Il y a deux grandes classes d'algorithmes de répartition dynamique :

— Dans les approches *à l'initiative de l'émetteur* [EAG 86] (sender initiated), les sites fortement chargés recherchent des sites moins utilisés pour y transférer des tâches locales.

— Dans l'approche *à l'initiative du récepteur* [LIV 82] (receiver initiated), les sites inutilisés cherchent à soulager les sites chargés en important certaines de leurs tâches.

Les deux sections suivantes présentent brièvement les stratégies de placement et de migration de GatoStar. Pour plus de détails le lecteur intéressé peut se référer à [FOL 95].

Ces deux techniques peuvent se combiner, il s'agit alors de l'approche *initiation symétrique* [FOL 95].

Les algorithmes peuvent s'appuyer sur un mécanisme de *placement initial* [ZHO 93]. Dans ce cas, les tâches sont allouées à leur création sur un site et y restent jusqu'à leur terminaison. Ils peuvent également se reposer sur la *migration* [BAR 89, BAS 99]. Dans ce cas, les tâches en cours d'exécution peuvent être suspendues et déplacées sur un nouveau site.

La plupart des algorithmes proposés utilisent des seuils de charge pour prendre leurs décisions de placement ou de migration. Ceci permet de minimiser les échanges d'informations tout en gardant des performances proches d'algorithmes plus complexes [ZHA 95].

2.2. *Placement*

La stratégie d'allocation de GatoStar est basée sur l'utilisation de seuils. Chaque programme est lancé sur la machine la moins chargée uniquement si la charge de celle-ci est inférieure au *seuil de surcharge*. Lorsque la charge d'un site dépasse la valeur du *seuil de surcharge*, celui-ci se retire temporairement du système de répartition. Ce seuil a un double emploi : ne pas saturer une machine et réduire les interférences avec les propriétaires des machines. La valeur du seuil, qui a été définie originellement en fonction d'autres études, a été fixée à 1,0 [EAG 86, ALO 88, BER 91](i.e., en moyenne la machine a toujours un processus prêt à s'exécuter). Elle reflète l'utilisation à 100 % des capacités d'une machine.

GatoStar intègre également des algorithmes multi-critères dont le but est de prendre en compte non seulement l'état des machines (et le respect des utilisateurs interactifs), mais également le comportement des applications en considérant les critères d'allocation donnés dans la description de l'application. L'état des machines est caractérisé par : la vitesse des processeurs, la vitesse des disques et des liens de communication, la mémoire disponible, le taux d'occupation des processeurs. Le comportement des applications est défini par : le temps processeur requis, la mémoire nécessaire à l'exécution, les fichiers utilisés, les communications entre programmes. Ces informations sont soit fournies directement par le programmeur d'application soit obtenues automatiquement par un mécanisme de trace intégré dans GatoStar. Le programmeur impose alors l'ordre des critères à prendre en compte.

2.3. *Migration de processus*

Les algorithmes de migration agissent après les algorithmes de placement en déplaçant les processus *en cours d'exécution* en fonction de l'évolution des applications et du système. L'évolution du système est due à des activités étrangères à GatoStar : un utilisateur se connecte ou bien envoie des processus sur des machines déjà chargées. L'évolution d'une application résulte des variations dans l'utilisation des ressources par les processus de l'application (occupation processeur, demande mémoire, communication entre processus).

La prise de décision est basée sur l'utilisation de deux seuils opposés : un *seuil de migration* et un *seuil de réception*. Sur chaque site un serveur local évalue périodiquement deux conditions :

— *Surcharge* : la machine locale est au-dessus du *seuil de migration* et au moins une machine est en dessous du *seuil de réception*. Dans ce cas, la machine locale est considérée comme émettrice et cherche à expulser des tâches.

— *Souscharge* : une machine est en dessous du *seuil de réception*. La machine devient alors réceptrice et cherche à importer des tâches.

Ces deux seuils permettent d'une part de soulager les machines fortement chargées et d'autre part de profiter des machines qui se libèrent. Dans tous les cas, la migration n'est décidée que si l'écart de charge entre les machines source et cible est significatif.

Il faut alors choisir un processus parmi les processus actifs dans le site émetteur. Tous les processus GatoStar ayant été exécutés localement pendant un temps minimum sont candidats à la migration. Ce temps minimum évite à un processus de passer son temps à être déplacé de machine en machine et empêche également les processus à courte durée de vie, d'être candidats à la migration.

GatoStar assure la transparence du mécanisme de migration. Les communications et accès aux fichiers se font par l'intermédiaire d'une bibliothèque spécifique qui reprend l'interface des appels système Unix pour les fichiers et une interface de type RPC pour les communications. Un service de nommage indépendant de la localisation et un mécanisme de suivi des communications permettent de masquer totalement la migration d'un processus au niveau applicatif.

3. Le modèle

À l'aide de l'outil de simulation QNAP2, nous avons développé un modèle générique d'un système de distribution en intégrant plusieurs algorithmes de distribution et plusieurs politiques d'échange d'informations entre les sites. Le passage d'un système à un autre se fait dans une phase de configuration du modèle en choisissant les algorithmes désirés et les valeurs de paramètres.

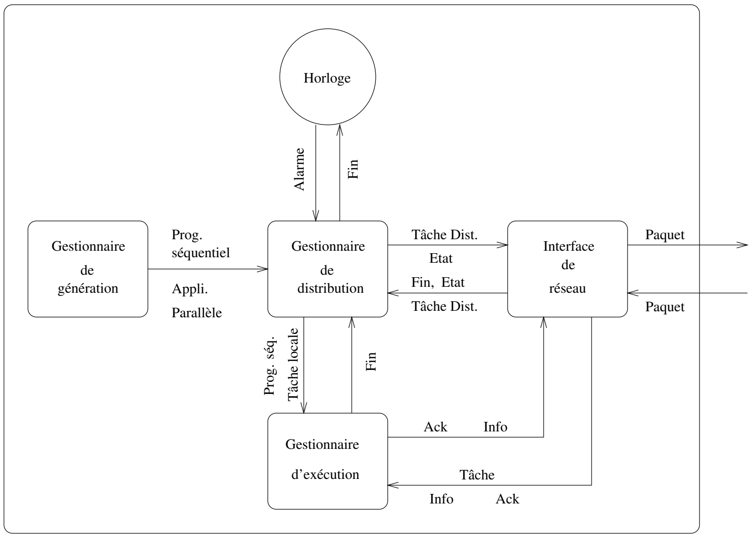

Figure 2. *Le modèle d'un site*

Le modèle est composé de cinq modules : l'horloge du site, le générateur de programmes, le gestionnaire de distribution, le gestionnaire d'exécution et l'interface de réseau. La figure 2 montre les transitions des clients entre les différentes composantes d'un site.

Les entités dynamiques (processus, messages, etc.) sont modélisées par des clients qui circulent dans les files d'attente du modèle. À chaque client est associé deux attributs principaux : sa priorité et sa classe (son type). La classe d'un client détermine l'algorithme à exécuter au niveau du serveur où se trouve ce client. Sa priorité détermine une priorité d'accès au serveur par rapport aux autres clients qui sont présents dans la même file.

Les sections suivantes décrivent les cinq modules composant le modèle.

3.1. *L'horloge*

Ce serveur simule une horloge dans le site. Il est constitué d'une source qui crée périodiquement un client et l'envoie au gestionnaire de distribution. Ce client est considéré comme une interruption de l'horloge ; il a donc une priorité élevée par rapport aux autres.

3.2. *Le gestionnaire de génération*

Le gestionnaire de génération est composé de deux sources de clients : les programmes séquentiels et les programmes parallèles. Un programme séquentiel est composé d'une seule tâche qui s'exécute sur le site local. Un programme parallèle est composé d'un ensemble de tâches reliées entre elles par des contraintes de précédence et de communication. Ces contraintes sont représentées par deux graphes : *graphe de précédence* et *graphe de communication*.

L'exécution d'un programme parallèle consiste à parcourir le graphe de précédence en largeur afin d'extraire les tâches qui peuvent être exécutées en parallèle, puis de les distribuer sur les sites disponibles.

Une tâche est un fragment de code exécuté séquentiellement. Ce fragment est divisé en trois composants : le premier (la partie calcul) est exécuté sur le processeur, le second représente les opérations d'entrée/sortie, le dernier définit les communications. Nous supposons que les opérations d'entrée/sortie et de communication sont dispersées dans la partie calcul. Pour les programmes séquentiels, la tâche contient uniquement les deux premiers composants. Un client sortant du gestionnaire de génération est envoyé au gestionnaire de distribution qui gère son placement.

3.3. *Le gestionnaire de distribution*

Le rôle du gestionnaire de distribution est d'une part de gérer la distribution des nouvelles tâches dans le réseau en utilisant un des algorithmes de placement et d'autre part de traiter les brusques variations de charge en appliquant un des algorithmes de migration. Il est composé de six serveurs : le *serveur de graphe*, le *serveur de placement*, le *serveur de mise à jour*, le *serveur de migration*, le *serveur de diffusion* et le *serveur de réception*.

1. Le **serveur de graphe** extrait les tâches exécutables du graphe d'exécution et les envoie au serveur de placement. Il reçoit deux types de clients. Le premier type représente les programmes parallèles avec leurs graphes de précédence envoyés par le gestionnaire de génération. Le deuxième type représente les messages générés par l'unité d'exécution qui indiquent l'achèvement et le résultat de l'exécution d'une tâche. Selon le cas, soit le serveur annonce la fin de l'exécution du programme, soit il déclenche des nouvelles tâches.

2. Le **serveur de placement** simule les algorithmes de placement du système de distribution de charge. Il est chargé de choisir et d'exécuter sur le «meilleur»site des tâches reçues du serveur de graphe, en tenant compte de l'état global du système et des critères de placement. Il envoie les tâches à exécuter à distance vers le gestionnaire de communication et les tâches à exécuter localement au serveur de mise à jour.

3. Sur chaque site, une variable locale *SystemState* représente l'état global du système vu par le site. Le rôle du **serveur de mise à jour** est de mettre à jour cette variable après la réception d'une tâche exécutée localement ou d'une tâche provenant d'un autre site.

4. Le **serveur de migration** gère le transfert des tâches entre les sites pour équilibrer la charge du réseau. Il reçoit périodiquement un client généré par l'horloge qui déclenche l'exécution d'un des algorithmes de migration décrits en 2.3

5. Le **serveur de diffusion** transmet périodiquement l'état du site à tous les sites du réseau. Le transfert se fait en utilisant soit la diffusion classique à tous les sites, soit une diffusion circulaire sur un anneau logique qui relie les différents sites du réseau. Le choix de la méthode de diffusion est précisé pendant la phase de configuration.

6. Le **serveur de réception** reçoit les messages d'états envoyés par les autres sites. Il met à jour sa vision de l'état des autres sites selon le contenu des messages reçus.

3.4. *Le gestionnaire d'exécution*

Le gestionnaire d'exécution est chargé d'exécuter les tâches arrivant sur le site. Il est composé de quatre modules: le processeur, le disque, le serveur de messages et le serveur de fin d'exécution. Les liens entre les différents serveurs sont détaillés dans la figure 3. Une tâche durant son exécution, est soit dans le processeur exécutant des opérations de calcul, soit bloquée dans une opération d'entrée/sortie (elle est dans la file du serveur disque) ou dans une opération de communication (elle est dans la file du serveur de messages). La tâche termine son exécution en se dirigeant vers le serveur fin d'exécution. Le pourcentage des opérations d'entrée/sorties est précisé par le paramètre P_{IO} et celui de la communication est précisé par le paramètre P_{comm}. Nous détaillons ci-dessous chacun des modules de l'unité d'exécution.

1. Le **processeur** représente les processeurs d'une machine et l'algorithme de l'ordonnancement. Il est composé de plusieurs files d'attente qui reflètent différents niveaux de priorité et de serveurs correspondant au nombre de processeurs dans la machine.
L'algorithme d'ordonnancement utilisé est celui du système Unix 4.4 BSD [MCK 96]. Cet algorithme est basé sur des files *multi-niveaux à rétroaction.* Tous les processus possèdent une priorité d'ordonnancement qui détermine la file d'exécution dans laquelle ils sont placés. Le nombre des files dans le simulateur est fixé à 3 (correspondant aux trois types de processus utilisés dans le modèle) au lieu de 32 dans 4.4BSD. La première file contient les tâches produites par les différents gestionnaires de distribution de charge. La deuxième file contient les tâches produites par les programmes séquentiels qui appartiennent à l'utilisateur local de la machine. La troisième file contient les tâches produites par les programmes parallèles exécutés à distance.

2. Le **disque** exécute la partie entrée/sortie du code de la tâche reçue. On distingue les accès locaux des accès distants. L'accès distant est naturellement

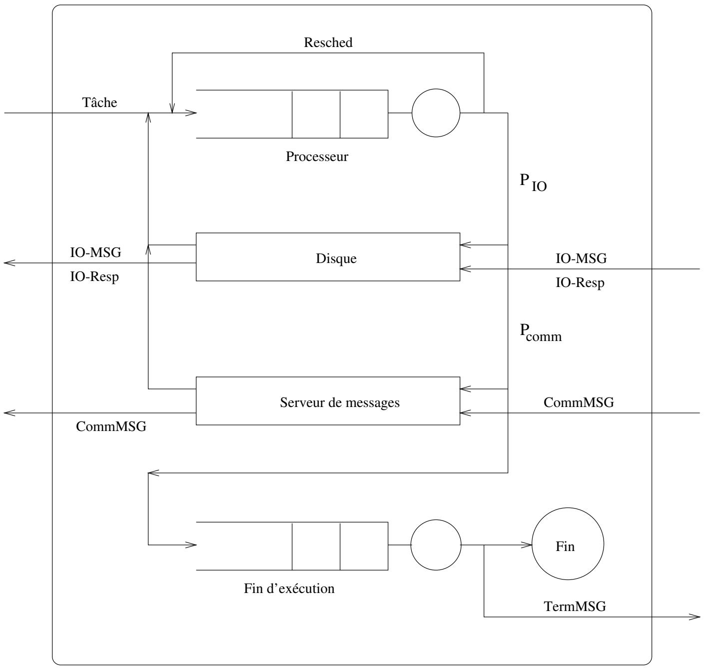

Figure 3. *Le gestionnaire d'exécution*

plus long que l'accès local et produit en plus une certaine charge au niveau du réseau.

3. Le **serveur de messages** exécute les primitives de communication entre les tâches de l'application. On distingue les communications synchrones et asynchrones.

4. À la fin de son exécution, en sortant du processeur, la tâche entre dans **le serveur fin d'exécution**. Si la tâche représente un programme séquentiel, le serveur envoie un message au gestionnaire de distribution local pour mettre à jour l'état du site. Si la tâche appartient à un programme parallèle, alors le serveur dirige le message au gestionnaire de distribution du site propriétaire (celui qui a lancé l'exécution du programme parallèle). Ce message représente les résultats des calculs effectués par la tâche. À la réception de ce message, le gestionnaire de distribution met à jour l'état du site et teste le graphe de dépendances du programme pour déclencher d'autres tâches. Le délai produit par ce serveur représente la phase de récupération des résultats et leurs mises sous forme d'un message.

3.5. *Le réseau*

Nous considérons le réseau comme un support logique qui réalise, d'une part, l'échange de messages entre les différentes tâches et, d'autre part, le transfert de tâches entre les différents sites. Nous distinguons trois politiques pour l'accès au réseau. Dans la première, un seul site a l'accès au réseau à un instant donné. Dans la deuxième, plusieurs sites ont le droit d'accès simultanément. Enfin dans la troisième, tous les sites ont simultanément le droit d'accès. Pour implémenter les trois politiques, nous avons utilisé une file d'attente équipée de plusieurs serveurs. Le temps de service de chaque serveur modélise le temps nécessaire pour envoyer un paquet. La longueur de la file d'attente représente la charge du réseau. Lors de la configuration du simulateur, la politique d'accès est précisée, ainsi que la distribution de temps de service et sa valeur moyenne.

3.6. *Estimation de la charge et surcoût des algorithmes*

Pour estimer la charge au niveau de chaque site dans le modèle, nous avons utilisé la formule du système Unix 4.4BSD [MCK 96]. En appliquant cette formule, nous avons une estimation de la charge identique à l'estimation utilisée dans le Unix 4.4BSD.

Il y a deux sources principales de surcoût dans un système de distribution de charge : l'exécution des algorithmes de distribution et l'échange des informations sur l'état du système.

Le surcoût introduit par l'échange des informations est pris en compte implicitement par le simulateur, car cet échange est réalisé par des transmissions et des réceptions de messages contenant les états du système. Ces messages subissent alors les différents délais des serveurs.

Le surcoût introduit par l'exécution des algorithmes de distribution est plus difficile à être simulé. La solution adoptée est basée sur l'idée que l'exécution de ces algorithmes prend une certaine capacité du processeur. Ainsi, à chaque activation d'un des serveurs du module de distribution, une tâche de priorité élevée est générée. Cette tâche, appelée *tâche système*, est toujours exécutée sur le processeur local. La distribution des durées d'exécution de chacune de ces tâches et la valeur moyenne sont précisées pendant la configuration.

3.7. *Validation et configuration*

La validation consiste à trouver les meilleures valeurs des différents paramètres du modèle pour lesquelles les résultats de simulations sont très proches des résultats du système réel. Cette phase permet de calibrer le simulateur.

La figure 4 illustre deux applications que nous avons exécutées sur le simulateur et sur le système réel. La figure 5 montre les résultats obtenus en

Application	Nombre de tâches	Durée d'une tâche
X	20	100 sec.
Y	12	1200 sec.

Figure 4. *Caractéristiques des applications X et Y*

fonction du nombre de sites. Nous avons observé des résultats similaires sur de nombreuses autres applications. La différence constatée entre les mesures réelles et les mesures de simulations varie entre 5 et 10 %.

Nous avons également calibré le simulateur en fonction des caractéristiques physiques de l'environnement d'exécution de GatoStar à savoir un réseau Ethernet à 10 Mbit/s et un temps moyen d'accès aux disques locaux de 10 ms. De même, nous avons introduit les paramètres de surcoûts des algorithmes qui correspondent à ceux observés sur GatoStar. De nombreuses comparaisons ont été faites afin de vérifier le simulateur. Ces études ont confirmé le bon comportement du modèle en particulier pour les communications et les accès au disque [HAJ 99]. À partir de ces résultats, nous avons estimé que ce modèle constitue une image proche du système réel et toutes les études présentées dans la section suivante ont été réalisées avec ce modèle.

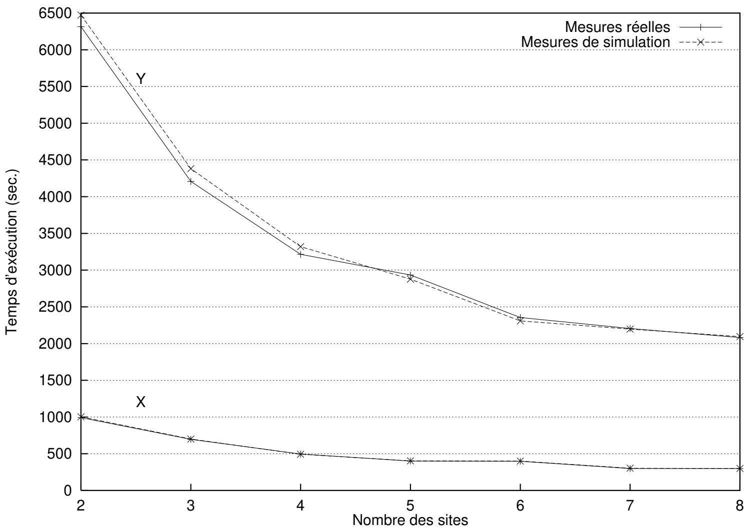

Figure 5. *Temps d'exécution de X et Y*

Application	N	T	durée totale
A	10	100	1000 sec.
B	20	250	5000 sec.
C	30	500	15000 sec.

Figure 6. *Description des applications testées*

4. Évaluation de performances

Dans cette section nous présentons, tout d'abord, une évaluation de performances des stratégies d'allocation. Puis, nous analysons l'impact de la topologie de l'application (en termes de granularité) sur les performances.

4.1. *Stratégie d'allocation*

Nous présentons les performances des algorithmes de placement du système de répartition en changeant les différents composants de la charge. La performance sera exprimée en gain par rapport à une exécution locale sur une machine. Le réseau simulé reproduit, en termes d'hétérogénéité de puissance, la configuration du réseau de notre laboratoire (cinq Sparcs 5 cadencée à 85 MHz, une Sparc 5 à 70 MHz, une Sparc 10 à 41 MHz et une Sparc 10 à 51 MHz)

4.1.1. *Placement*

Dans un premier temps, nous supposons que toutes les machines sont oisives et qu'aucun utilisateur n'est connecté à sa machine. Nous supposons aussi qu'une seule application de calcul intensif est lancée dans le système pour éviter l'interférence entre les applications. Cette application est composée de N tâches homogènes, chacune de durée T secondes. Il n'y a pas de contraintes de précédences entre les tâches, elles peuvent donc toutes s'exécuter en parallèle. La figure 6 montre les valeurs de N et de T pour les trois applications testées.

La figure 7 montre le gain pour les trois applications en fonction du nombre de sites. On remarque que le gain maximum pour l'application A est autour de 10, il est atteint à partir de 15 sites car l'algorithme de placement choisit les machines les plus rapides pour placer les 10 tâches de l'application. Ce résultat est dépendant de l'hétérogénéité du réseau simulé : dans notre configuration au-dessus de 10 machines le nombre de machines puissantes reste faible. Une étude plus détaillée sur l'impact de l'hétérogénéité est présentée en 4.4. Le gain maximum pour l'application B est autour de 19, il est atteint pour 30 sites et celui de l'application C est autour de 21, il est atteint à partir de 30 sites.

On remarque aussi une diminution des performances lorsque le nombre de sites dépasse un certain nombre ; ceci est principalement dû au surcoût des

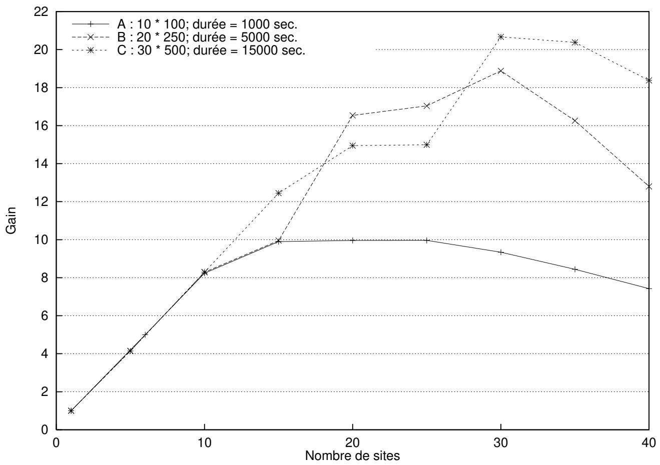

Figure 7. *Gain de l'algorithme de placement*

algorithmes de distribution. Ainsi, les algorithmes de placement ont une efficacité optimum si le nombre de machines du réseau est inférieur à 25 pour l'application A, à 30 pour B et à 35 pour C. Nous en déduisons que la limite d'efficacité de ces algorithmes dépend de la configuration de charge à exécuter et la configuration de l'environnement. Ainsi, des informations sur le comportement des applications (nombre de tâches, durées, ...) peuvent améliorer les performances des algorithmes en permettant d'ajuster dynamiquement leurs paramètres, comme nous le verrons par la suite.

Maintenant, nous réalisons une nouvelle série de mesures en ajoutant une charge locale à chaque machine. Cette charge locale simule la charge induite par l'utilisateur local de la machine.

Pour simplifier la configuration de la charge locale de chaque machine nous supposons que l'arrivée des programmes est poissonnienne de taux λ et le temps d'exécution d'un programme est de distribution exponentielle de moyenne μ[1]. λ et μ sont des paramètres. Nous étudions les performances en fonction du niveau de la charge locale. Donc, pour des valeurs faibles de $\rho = \frac{\lambda}{\mu}$ les machines sont peu chargées par leurs utilisateurs ; pour des valeurs de ρ proches de l'unité, le système devient très chargé. Nous supposons que les tâches générées par les utilisateurs locaux sont prioritaires par rapport aux tâches appartenant aux programmes parallèles, ainsi l'utilisateur sera peu gêné par les tâches étrangères. La figure 8 détaille trois applications testées.

La figure 9 montre le gain de performances des trois applications exécutées sur un réseau de 4 machines en fonction de niveau de la charge locale. On remarque que le placement des tâches parallèles reste rentable jusqu'à une

1. Nous avons estimé que le paramètre le plus significatif est la charge locale moyenne et non la loi d'arrivée.

Application	N	$T(sec.)$
$A0$	20	10
$A1$	20	50
$A2$	20	250

Figure 8. *Valeurs de N et de T des trois applications testées*

certaine valeur de la charge locale que nous appelons *niveau local maximum* (1,2 pour l'application $A0$, 1,5 pour l'application $A1$ et 1,7 pour l'application $A2$). On remarque que plus le temps d'exécution est grand, plus le niveau maximum est grand.

Nous avons également étudié l'impact du nombre de machines et remarqué que plus le nombre de machines est important, plus le placement reste efficace dans des conditions de charge élevées.

Nous déduisons de ces deux études que le niveau maximum de la charge locale dépend à la fois de l'application et de l'environnement. Savoir caractériser le niveau maximum pour une application donnée et pour un réseau donné est très utile pour les algorithmes de placement, car au-delà de ce niveau, on sait que les placements distants sont inutiles et qu'il est préférable d'exécuter localement les tâches parallèles ou de les retarder.

Des tests semblables à ceux du paragraphe précédent ont été faits pour étudier l'influence des opérations d'entrées/sorties sur les performances. Les résultats de ces tests montrent que le gain est acceptable pour un pourcentage d'entrée/sortie inférieur à 35%. Au-delà, le gain devient très petit par rapport au nombre de machines utilisées.

Pour comparer à la fois l'influence des opérations de communications et

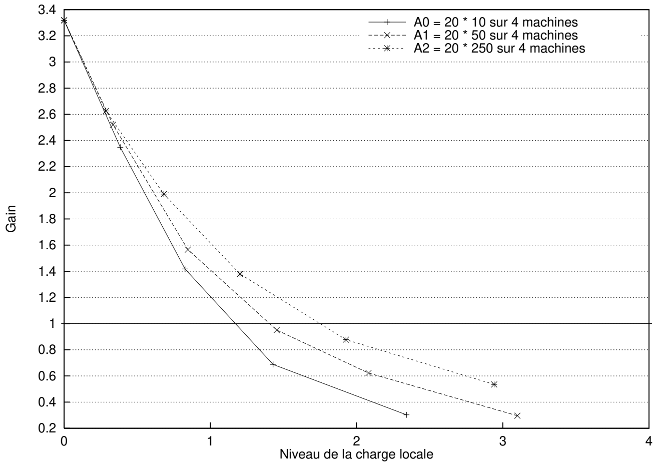

Figure 9. *Dépendance de la charge locale*

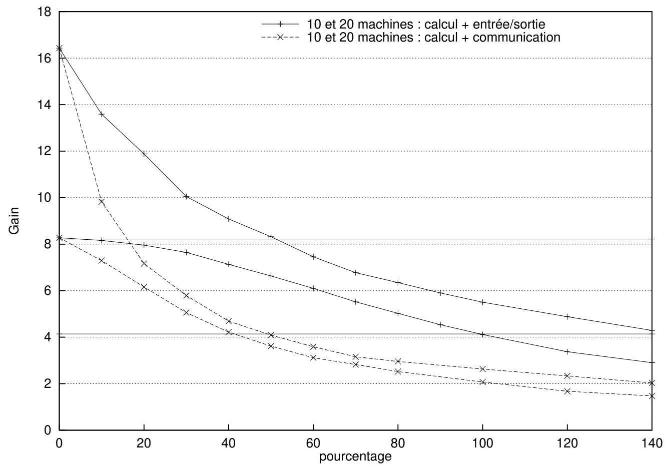

Figure 10. *Impact des communications et entrées/sorties*

des opérations d'entrée/sortie, nous avons choisi deux applications chacune constituée de 20 tâches. Chaque tâche de la première application est composée d'une partie calcul et d'une partie entrée/sortie ; chaque tâche de la deuxième application est composée d'une partie calcul et d'une partie communication. Les deux applications ont été exécutées sur un réseau de 10 et 20 machines.

La figure 10 montre le gain de performances en fonction du pourcentage d'entrée/sortie pour la première application et du pourcentage de communication pour la seconde. On remarque que les gains de la première application sont plus élevés que ceux de la seconde ce qui indique que les communications ont une influence plus forte sur les performances que les entrées/sorties. Par exemple, le gain est divisé par 2 à partir de 50 % d'E/S, et à partir seulement de 20 % de communications. Nous en déduisons pour les algorithmes de placement que le critère de communication est prioritaire par rapport au critère d'entrée/sortie.

4.1.2. *Migration*

Nous examinons maintenant le gain supplémentaire apporté par la migration par rapport au placement. Nous avons considéré les trois applications homogènes A, B et C présentées dans la figure 6.

La figure 11 montre la comparaison des gains entre une stratégie de placement seul et une autre combinant placement et migration. On remarque que les performances sont quasi-identiques pour A et B (en faveur du placement seul à cause du surcoût supplémentaire de la migration). Cela s'explique car, dans ce cas, le placement effectué est proche du placement optimal. À cause de l'homogénéité des tâches et des différences mineures de puissance des machines, le transfert de tâches vers les machines rapides libérées ne suffit pas pour compenser le surcoût de la migration.

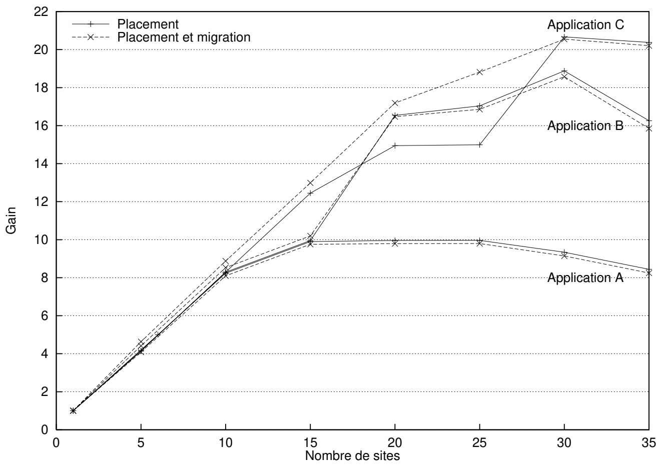

Figure 11. *Gain de migration pour les applications homogènes*

L'application C, qui est composée de 30 tâches, se comporte différemment entre 15 et 30 machines. Par exemple, pour 20 machines, le placement initial est le suivant : deux tâches sur les dix machines les plus rapides et une tâche sur les dix autres. Après un certain temps, les 10 dernières tâches terminent leur exécution, à ce moment la migration d'une des deux tâches des machines rapides vers une machine plus lente mais inutilisée améliore fortement les performances. Le même raisonnement est applicable dans le cas de 25 machines. Le gain supplémentaire pour 20 machines est de l'ordre de 15 %, et de 26 % pour 25 machines. Le gain supplémentaire dans le cas de 25 machines est justifié car les 5 machines recevant les tâches transférées sont les 5 machines les plus rapides.

D'autres études ont été faites pour des applications irrégulières avec des tâches de durée variable. On constate là aussi des gains importants dans certaines configurations dus au même phénomène observé pour l'application C. Des résultats précédents, nous caractérisons les deux cas pour lesquels la migration peut ajouter un gain supplémentaire conséquent :

— les tâches sont homogènes mais la structure de l'application et le nombre des sites disponibles amènent forcement à un état de déséquilibre (par exemple, l'application C avec 20 et 25 sites).

— il existe un degré élevé d'hétérogénéité de temps d'exécution des tâches pouvant générer des états de déséquilibre dans le système.

Dans les deux cas précédents, et selon tous les tests effectués sur la migration nous n'avons pas pu observer un gain supplémentaire dépassant 35 % (de nombreuses autres distributions mathématiques ont été testées avec différents degrés d'hétérogénéité). Nous avons également mesuré les temps de réponses en appliquant une stratégie de migration seule (sans placement initial) comme cela a été fait dans le système MOSIX [BAR 89]. Dans tous les cas, les gains de performances observés étaient inférieurs aux deux stratégies présentées.

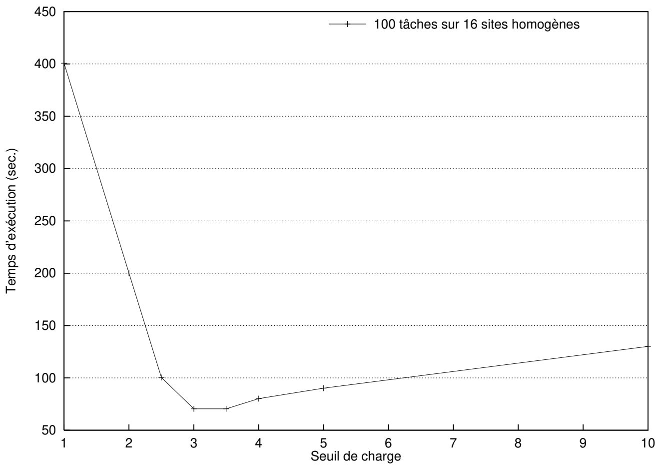

Figure 12. *Variation de temps d'exécution en fonction du seuil de charge*

4.2. *Adaptabilité des algorithmes de distribution*

Nous montrons dans cette partie, la nécessité d'adaptation des algorithmes de distribution de charge. Notre étude porte sur la valeur des seuils qui sont à la base des algorithmes de prise de décision.

Pour comprendre l'influence des valeurs des différents seuils sur les performances du système, nous avons désactivé les algorithmes de migration et étudié l'influence de la valeur du *Seuil de charge* sur les performances des algorithmes de placement uniquement.

La figure 12 montre la variation de temps d'exécution en fonction du *Seuil de charge* pour une application de 100 tâches exécutée sur 16 machines. On remarque que le temps d'exécution commence par une valeur maximum (correspondant au placement local de toutes les tâches), puis rapidement décroît jusqu'à une valeur minimum (nous appelons la valeur de seuil correspondante *le seuil optimal*). Nous en déduisons que le choix de la valeur de seuil est très important.

À partir du résultat précédent, nous avons simulé deux stratégies de seuil. Dans la première stratégie, implémentée dans GatoStar, l'allocation et la migration sont basées sur des seuils fixes. Dans la deuxième, les seuils sont adaptatifs et dépendent de la charge globale du système. Les différents seuils utilisés sont définis de la manière suivante :

$Seuil\ de\ charge = charge\ moyenne$

$Seuil\ de\ reception = (1 - LoadFactor) * charge\ moyenne$

$Seuil\ de\ migration = (1 + LoadFactor) * charge\ moyenne$

où $LoadFactor$ est expérimentalement fixé à 0.4. La charge moyenne est calculée périodiquement avec la formule suivante :

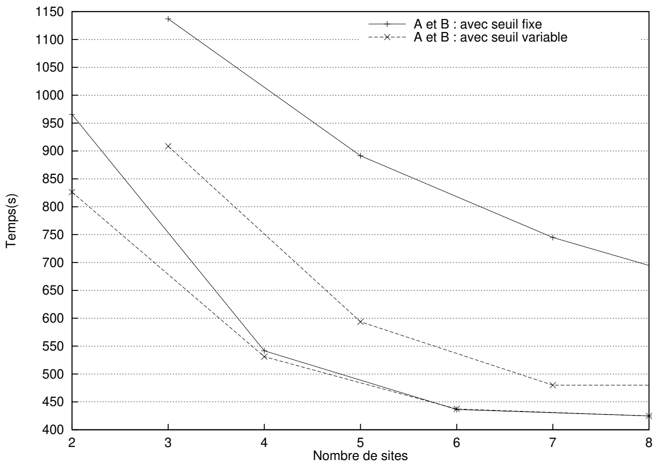

Figure 13. *Simulation de temps de réponse selon la politique de seuil*

$$charge\ moyenne = \frac{\sum_{i=1}^{N_p} \frac{LoadAvg_i}{vitesse_i}}{Nombre\ de\ machines}$$

Dans cette politique, les valeurs des seuils suivent toujours la charge moyenne du système.

La figure 13 présente une comparaison de ces deux politiques pour deux applications parallèles : l'application A est composée de 40 tâches identiques, l'application B est composée de 48 tâches de durées différentes (24 tâches longues et 24 tâches courtes). On remarque que la politique adaptative à seuils variables améliore le temps de réponse du système particulièrement pour les applications hétérogènes (jusqu'à 30 % pour l'application B).

A partir de ces résultats, la stratégie des seuils adaptatifs a été intégrée dans le système GatoStar. Des améliorations importantes des performances ont été effectivement observées sur des applications réelles.

4.3. *Impact de la granularité*

Il y a deux critères importants qui caractérisent les algorithmes de distribution. Le premier est le gain apporté par ces algorithmes et le deuxième est le surcoût induit par l'utilisation de ces algorithmes. Il est difficile de trouver un compromis entre les deux critères dans un environnement dynamique. Nous examinons les performances des algorithmes de distribution en vue de comprendre la relation entre la charge du système (les applications) et les deux critères des algorithmes de distribution.

Nous considérons une application parallèle ayant une masse de calcul de D

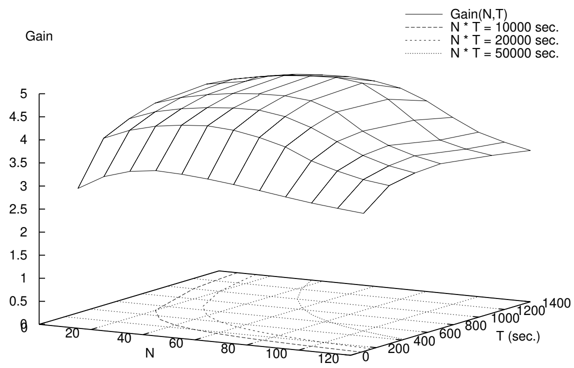

Figure 14. *Gain en fonction de la granularité*

secondes. Nous supposons aussi que cette masse de calcul est divisible en N tâches chacune est de durée T secondes. Pour simplifier le raisonnement, nous supposons aussi que D est égal à N multiplié par T.

La figure 14 indique le gain en fonction de N et T et montre la projection des trois courbes de D pour les valeurs 10000, 20000 et 50000 secondes. Nous trouvons ainsi que pour 10000 secondes le N optimum est entre 9 et 12 tâches et le T optimum est entre 900 et 1100 secondes. Pour 20000 secondes, N doit être compris entre 17 et 20 et T entre 1000 et 1200 secondes. Pour 50000 secondes, N doit être compris entre 65 et 80, T entre 600 et 800 secondes. En effet, si N est trop petit l'application perd trop de parallélisme pour présenter un gain important. En revanche pour N grand, les tâches sont trop petites pour que les stratégies de placement et de migration soient vraiment rentables.

4.4. *Hétérogénéité du réseau*

Pour étudier l'influence de l'hétérogénéité sur les performances, nous avons introduit la notion de vecteur de puissance. Le vecteur de puissance d'un réseau de M machines est un vecteur de M composantes où la composante i représente la vitesse de la machine numéro i. Par exemple, le vecteur de puissance pour l'environnement du laboratoire est :

$$G = (1.205, 1.205, 1.205, 1.000, 1.205, 1.024, 1.205, 1.265)$$

Nous exprimons la puissance d'un réseau en définissant la norme donnée par :

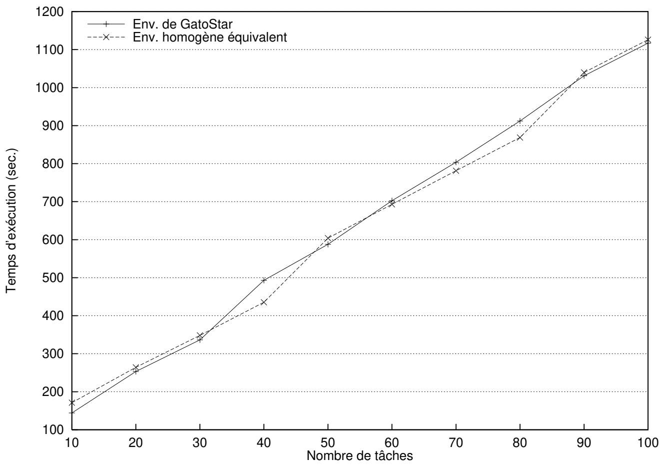

Figure 15. *GatoStar sur un environnement homogène*

$$\|P\| = \sum_{i=1}^{M} p_i$$

où p_i est la vitesse de la machine i. On dit que deux réseaux sont équivalents si leurs vecteurs de puissance ont la même norme.

Nous avons comparé les performances du système GatoStar dans deux environnements : l'environnement du laboratoire et l'environnement homogène équivalent. Nous considérons une application homogène de N tâches de durée T secondes. La figure 15 montre le temps d'exécution en fonction de N. On remarque que la différence entre les performances de l'environnement du laboratoire et l'environnement homogène équivalent est négligeable. Nous en déduisons que la norme définie ci-dessus peut être utilisée pour mesurer la puissance du réseau.

5. Autres approches

Les modèles de distribution de charge proposés dans la littérature n'intègrent généralement pas le niveau de détails de notre modèle. Des approximations importantes sur les lois d'arrivée, le temps de transfert des tâches et leur distribution ont tendance à éloigner les performances simulées du comportement réel. L'exemple le plus connu est celui du modèle d'Eager [EAG 89] qui concluait en 1989 que la migration ne pouvait apporter aucun gain significatif sur un réseau de stations de travail. Des expérimentations ont montré depuis que cette conclusion est fausse. Harchol-Balter et Downey montrent dans [HAR 97] que ceci est en partie dû à l'hypothèse d'une loi d'arrivée de tâche exponentielle.

Lin et Raghavendra [LIN 91] utilise une version modifiée du modèle $M/M/n$ pour étudier l'efficacité d'un modèle centralisé de distribution de charge. Ils supposent que l'arrivée des processus est poissonnienne de taux λ. Le temps de service de chaque processus est de distribution exponentielle de moyenne $\frac{1}{\mu}$; il est donc indépendant de la taille et la durée de la tâche ce qui n'est toujours le cas dans la réalité. Ils supposent également que le délai d'échange des informations entre les sites est négligeable.

Evans et Butt [EVA 93] ont développé un réseau de files d'attente pour étudier les performances d'un modèle réparti de distribution de charge. Le modèle est très simplifié car chaque site du réseau est composé uniquement de deux files d'attente. La première représente les algorithmes d'équilibrage de charge et la deuxième représente l'unité d'exécution. Le temps de service de la première station est supposé négligeable et celui de la deuxième est de distribution exponentielle. De plus, la migration n'est pas prise en compte.

Harchol-Balter et Downey [HAR 97] ont particulièrement étudié la distribution des tâches en mesurant les durées de vie d'un grand nombre de processus (plus d'un million) dans les milieux académiques. Ils ont trouvé que la distribution de la durée de vie des processus n'est pas exponentielle ou hyper-exponentielle mais qu'elle coïncide avec une courbe de forme T^k, avec $-1.3 < k < -0.8$. Ils ont également constaté que 50 % de la charge des machines était due à moins de 1 % des processus. Ces processus correspondent à ceux ayant une vie de plus de 4 secondes. Leland et Ott [LEL 86] ont mené une étude similaire sur le comportement des tâches séquentielles dans les domaines commerciaux.

6. Conclusions

Nous avons présenté un modèle paramétrable et configurable de système de distribution de charge dans un réseau de stations de travail. Le modèle a été construit à partir d'un système de placement développé au laboratoire du LIP6 (GatoStar). L'avantage principal de notre approche, par rapport à d'autres modélisations, est de se rapprocher le plus finement possible des mesures observées dans un système réel de distribution de charge. Le travail présenté a nécessité des raffinements successifs des paramètres du modèle et du modèle lui-même jusqu'à obtenir des performances par simulation quasiment identiques à celles obtenues par le système GatoStar.

Nous avons mis en évidence l'efficacité des algorithmes de placement notamment pour les applications homogènes. Nous avons aussi étudié les performances des algorithmes de placement sur un réseau chargé par des utilisateurs locaux. Nous avons étudié l'influence des opérations d'entrées/sorties et des opérations de communication sur les performances. Nous avons trouvé qu'un critère basé sur la communication est plus prioritaire qu'un critère basé sur les entrées/sorties.

Nous avons également mis en évidence le surcoût et le gain apporté par de la migration par rapport aux algorithmes de placement. Nous en avons déduit deux points essentiels concernant ces algorithmes. Le premier est de savoir caractériser les applications où la migration peut être rentable. Le deuxième est de pouvoir détecter, pendant l'exécution de ces applications, le moment où il faut déclencher la migration.

Enfin, nous avons montré que l'efficacité d'un algorithme de distribution dépend en grande partie de sa capacité à s'adapter en fonction de la charge du système et de l'environnement d'exécution.

7. Bibliographie

[ALO 88] ALONSO R., COVA L. L., «Sharing jobs among independently owned processors », *The 8th International Conference on Distributed Computing Systems*, p. 365-372, 1988.

[BAR 89] BARAK A. , WHEELER R. , «MOSIX: An integrated multiprocessor unix », In *Proceedings of USENIX Winter 89*, p. 101–112, 1989.

[BAS 99] BASNEY J. , LIVNY B. , «Deploying a high thoughput computer computing », In *High Performance Cluster Computing*, p. 116–134, Prentice Hall, 1999.

[BER 91] BERNARD G., SIMATIC M., « A decentralized and efficient algorithm for network of workstations », *The European Conference for Open Systems Spring-91*, p. 139-148, 1991.

[HAJ 99] HAJMAHMOUD Y. , «Modélisation et évaluation de performances des systèmes de distribution de charge en environnements répartis », *Thèse de doctorat de l'Université Paris 6*, Mars, 1999.

[HAR 97] HARCHOL-BALTER M., DOWNEY A. B., «Exploiting process lifetime distributions for dynamic load balancing », *ACM Transactions on Computer Systems*, vol. 15, n° 3, 1997.

[MCK 96] MCKUSICK M. K., BOSTIC K., KARELS M. J.QUARTERMAN J. S., « The design and implementation of the 4.4BSD operating system », ADDISON-WESLEY, 1996.

[EAG 86] EAGER D. L., E. D. LAZOWSKA, ZAHORJAN J., «Adaptive load sharing in homogeneous distributed systems », *IEEE Trans. on Software Engineering*, vol. 12, n° 5, p. 662–675, 1986.

[EAG 89] EAGER D. L., E. D. LAZOWSKA, ZAHORJAN J., «Speedup versus efficiency in parallel systems », *IEEE Transactions. on Computers*, vol. 8, n° 3, p. 408–423, 1989.

[EVA 93] EVANS D. J., BUTT W. U. N., «Dynamic load balancing using task-transfer probabilities », *Parallel Computing*, vol. 19, p. 897–916, 1993.

[FOL 95] FOLLIOT B., SENS P., RAVERDY P-G., «Plate-forme de répartition de charge et de tolérance aux fautes pour applications parallèles en environnement réparti », *Calculateurs Parallèles*, vol. 7, n° 4, p. 345–366, 1995.

[LEL 86] LELAND W., OTT T., «Load balancing heuristics and process behavior », *ACM Performance Evaluation*, vol. 14, p. 54–69, 1986.

[LIN 91] LIN H., RAGHAVENDRA C. S., «A dynamic load balancing policy with a central job dispatcher (LBC) », *IEEE Transactions on Software Engineering*, vol. 18, n° 2, p. 264–271, 1991.

[LIV 82] LIVNY M. , MELMAN M., «Load balancing in homogeneous broadcast distributed systems », *ACM Performance Evaluation*, vol. 11, n° 1, p. 47–55, 1982.

[SEN 98] SENS P., FOLLIOT B., «The STAR fault manager for distributed operating environments. Design, implementation and performance », *Software-Practice and Experience*, vol. 28, n° 10, p. 1079-1099, 1998.

[TAN 85] TANTAWI A. N. , TOWSLEY D. F., «Optimal static load balancing in distributed computer systems », *Journal ACM*, vol. 32 p. 445–465, 1985.

[ZHA 95] ZHANG Y. , HAKOZAKI K. , KAMEDA H. , SHIMIZU K., «A performance comparison of adaptive and static load balancing in heterogeneous distributed systems », In *28th Annual Simulation Symposium*, p. 332–340, 1995.

[ZHO 93] ZHOU S. , ZHENG X. , WANG J. , DELISLE P. , «Utopia: a load sharing facility for large, heterogeneous distributed computer systems », *Software-Practice and Experience*, vol. 23, n° 12, p. 1305–1336, 1993.

Yanal Hajmahmound *a obtenu sa thèse de doctorat en Informatique de l'Université Paris VI, Pierre et Marie Curie en 1999. Ses recherches effectuées au Laboratoire d'Informatique de Paris VI (LIP6), associé au CNRS, portent sur le placement d'applications parallèles sur des réseaux de stations de travail. Ses centres d'intérêt sont la simulation et l'évaluation de performance de systèmes répartis.*

Bertil Folliot *est Professeur en informatique à l'Université Paris VI et Conseiller scientifique à l'INRIA-Rocquencourt. Ses activités de recherche incluent le placement dynamique et la tolérance aux fautes sur des réseaux de stations de travail, et plus récemment une plate-forme adaptative multi-langue : la machine virtuelle virtuelle.*

Pierre Sens *est Maître de Conférences à l'Université Paris VI. Ses recherches actuelles portent sur le placement multi-critère de tâches et la tolérance aux fautes sur des réseaux de stations de travail. Il s'intéresse également aux mémoires partagées réparties et plus particulièrement aux problèmes liés à leur extension sur plusieurs réseaux.*