Répartition de charge dans Stardust : un environnement pour l'exécution d'applications parallèles en milieu hétérogène

Gilbert Cabillic, Isabelle Puaut

Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA)
Campus Universitaire de Beaulieu
35042 RENNES Cédex
E-mail : {cabillic,puaut}@irisa.fr

RÉSUMÉ. *Stardust est un environnement pour l'exécution d'applications parallèles au-dessus d'un réseau de machines hétérogènes, comprenant à la fois des stations de travail et une machine parallèle. Stardust permet de construire des applications composées de plusieurs processus, communiquant à la fois par envois de messages et par mémoire virtuelle partagée. Par ailleurs, Stardust inclut un mécanisme permettant de capturer l'état des applications. Cet état peut alors, soit être sauvegardé sur disque dans le but de tolérer les défaillances, soit être transféré d'un ensemble de machines vers un autre pour équilibrer les charges des machines du système. Cet article donne une vue d'ensemble de Stardust en détaillant plus particulièrement son algorithme de régulation de charge.*

MOTS-CLÉS : *Programmation parallèle, Mémoire virtuelle partagée, Hétérogénéité, Répartition de charge*

1. Introduction

L'utilisation de réseaux de machines pour l'exécution d'applications parallèles apparaît comme une solution attrayante, complémentaire de celle des machines massivement parallèles. En effet, les réseaux de stations de travail, de plus en plus répandus, offrent une puissance de calcul cumulée comparable à celle des machines parallèles pour un coût comparativement moins élevé. De plus, des études récentes montrent que même pendant les horaires de travail, le taux d'inactivité des stations d'un réseau atteint en moyenne 90% [3]. De nombreux environnements ont été conçus pour tirer profit de la puissance potentielle des réseaux de stations pour l'exécution d'applications parallèles. Les propriétés souhaitables de tels environnements sont les suivantes :

– Gestion de l'hétérogénéité de l'architecture, permettant d'utiliser au sein d'une même application des machines de types différents, et ainsi d'utiliser toute la puissance de calcul disponible ;
– Tolérance aux fautes, permettant en cas de défaillance matérielle (ou arrêt logiciel d'une machine) de ne pas réexécuter entièrement les applications ;
– Régulation de charge, permettant de tirer profit des machines inutilisées, et de ne pas surcharger les stations en cours d'utilisation.

Stardust [2] est un environnement réunissant ces trois propriétés. Il permet l'exécution d'applications parallèles au-dessus d'un réseau de machines hétérogènes (stations de travail de type compatibles PC et machine parallèle Paragon d'Intel), et inclut des mécanismes de tolérance aux défaillances et de répartition de charge. Le paragraphe 2 donne une vue d'ensemble de Stardust. Le lecteur trouvera un exposé plus complet des fonctionnalités de Stardust dans le rapport [2]. Le partage de charge mis en place dans Stardust est détaillé dans le paragraphe 3. La réalisation et les performances de Stardust sont abordées dans le paragraphe 4. Enfin, le paragraphe 5 compare Stardust avec quelques travaux apparentés et conclut.

2. Vue d'ensemble de Stardust

Les processus d'une application développée avec Stardust peuvent s'exécuter sur des architectures de type différent. Cet aspect est décrit dans le paragraphe 3.1. Par ailleurs, Stardust inclut un mécanisme de capture de l'état des applications. Ce dernier, qui est décrit dans le paragraphe 3.2, est utilisé pour tolérer les défaillances et équilibrer la charge des machines du système.

2.1. *Gestion de l'hétérogénéité dans Stardust*

Les stations de travail reliées par un réseau sont souvent hétérogènes de par leur architecture matérielle et/ou logicielle.

Hétérogénéité matérielle

Les différences matérielles entre machines impliquent que la taille et l'agencement en mémoire des types de données (même les types élémentaires comme les entiers) peuvent varier d'un type d'architecture à un autre. Il en est de même pour la taille des pages virtuelles dans les systèmes supportant la pagination. Par exemple, la taille des pages sur la machine Paragon d'Intel est de 8Ko, alors qu'elle est de 4Ko sur des machines compatibles PC.

Pour supporter l'hétérogénéité matérielle, Stardust utilise un format de données standard (format XDR) pour les transmissions de données entre architectures de type différent. La transformation des données lors de leur transmission est effectuée sans intervention du programmeur, qui spécifie uniquement le type des données destinées à être échangées entre machines. Les transferts de pages, découlant du maintien de la cohérence des données dans la mémoire virtuelle partagée de Stardust, utilisent une taille de page standard (notion de page *hétérogène*). Par exemple, si les représentations des types élémentaires (entiers, valeurs flottantes) sont identiques pour plusieurs architectures, la taille d'une page hétérogène est le plus petit commun multiple (PPCM) entre les tailles de pages de toutes les architectures.

Hétérogénéité logicielle

Du point de vue logiciel, les systèmes d'exploitation s'exécutant sur les machines du système peuvent être différents, ce qui implique l'utilisation potentielle de protocoles de communication variés.

Un protocole de communication standard (TCP/IP) est utilisé pour les communications entres architectures de types différents. Dans un souci d'efficacité, ce protocole, ainsi que le format XDR, ne sont utilisés que pour les communications inter-architecture. Au sein d'une architecture donnée, et ceci grâce à la structuration interne de Stardust, aucune conversion de donnée n'est nécessaire. Le protocole de communication le plus efficace de l'architecture (par exemple NX sur la machine Paragon) est alors utilisé lors des communications entre sites d'une même architecture.

2.2. *Capture de l'état des applications*

Stardust offre un mécanisme pour capturer l'état des applications. Son principe de base consiste à sauvegarder l'état de chaque processus d'une application lorsque tous ses processus sont synchronisés sur une barrière de synchronisation, et uniquement dans leur programme principal (fonction C *main*). Ainsi, l'état de l'application constitue un état global cohérent [4]. Pour limiter le volume des informations à sauvegarder, et éliminer les informations dépendantes du type d'architecture et du système d'exploitation utilisé, seules les données stockées dans des régions de mémoire virtuelle partagée sont mémorisées. Ainsi, toutes les informations sauvegardées peuvent l'être dans un format indépendant machine, ce qui permet la migration d'une application entre machines de types différents (migration hétérogène). Dans la réalisation actuelle de Stardust, ce support pour la migration hétérogène peut toutefois impliquer un effort supplémentaire pour le programmeur dans le cas où ce dernier utilise des outils liés à un système d'exploitation donné (chiens de garde, fichiers, etc). Cette limitation peut être éliminée en implantant dans Stardust une interface commune, indépendante du système cible, pour ces différents outils.

La capture de l'état des applications est réalisé au moyen d'un processus par type d'architecture (CS, pour *Capture Server*). Dans la réalisation actuelle, le processeur 0 de la machine Paragon est utilisé, ainsi qu'une machine PC fixée au préalable. Chaque CS stocke l'état des applications sur un support de stockage insensible aux défaillances (système de gestion de fichiers partagé), et conserve en mémoire vive un cache de cet état. Chaque CS stocke les données en utilisant le format interne de l'architecture sur lequel il s'exécute, les données échangées entre CSs utilisent en revanche le format externe XDR. L'état d'une application peut être utilisé pour reprendre le calcul après une défaillance [1], ou équilibrer la charge des machines (cf § 3).

1. Ce point n'est pas détaillé dans l'article ; le lecteur est invité à consulter [1] pour plus de détails.

3. Régulation de charge dans Stardust

Un algorithme de répartition de charge est classiquement divisé en trois composants [7] : (i) *Une politique d'information*, qui spécifie la nature des informations de charge qui sont échangées entre machines, et la manière dont cette information est propagée ; (ii) *Une politique de transfert*, qui décide, à partir des informations de charge propagées par la politique d'information, quels sont les processus éligibles pour la régulation de charge (pour un placement initial ou une migration); (iii) *Une politique de placement*, qui détermine pour chaque processus élu par la politique de transfert, la machine sur laquelle le processus doit être transféré. Les principes de fonctionnement de ces trois politiques dans Stardust sont décrits ci-après.

3.1. *Politique d'information*

La définition d'une politique d'information dans un environnement hétérogène nécessite la sélection d'une mesure de charge commune à tous les types d'architecture. Dans Stardust, la charge d'une machine est représentée par un entier compris entre la valeur 0, représentant une machine inactive, et la valeur 100, représentant une machine très chargée. La définition de la charge d'une machine dépend du type de cette architecture. Sur la machine Paragon, pour laquelle le système d'exploitation supporte l'exécution d'un unique processus par processeur, la charge d'un processeur peut uniquement prendre la valeur 0 ou 100. En revanche, pour les machines de type PC, pour lesquelles le système d'exploitation permet l'exécution de plusieurs processus, toutes les valeurs de charge sont utilisées ; la valeur de la charge d'une machine dépend à la fois de charge du processeur et de celle de la mémoire de la machine.

La propagation de la charge repose sur un système hiérarchique. Un processus par type d'architecture (LIM, pour *Load Information Manager*) mémorise les charges de toutes les machines de cette architecture. Ce processus est dans la réalisation actuelle placé sur un site prédéterminé du système (processeur 0 sur la machine Paragon, machine d'adresse Internet donnée pour le réseau de PCs). Chaque LIM met à jour sa structure de données interne en interrogeant à intervalle fixe chaque machine de l'architecture dont il est responsable.

3.2. *Politique de transfert*

Le rôle de la politique de transfert est d'identifier les processus devant faire l'objet d'un placement initial ou d'une migration. Dans Stardust, cette politique considère globalement tous les processus constituant une application [2]. La politique de transfert repose sur un classement des applications par priorités. Dans le cas de surcharge globale des machines du système, seules les applications prioritaires seront exécutées; les autres applications seront suspendues jusqu'à ce que la charge du système atteigne à nouveau une valeur raisonnable. La politique de transfert est activée dans les trois

2. Par abus de langage, nous employons par la suite le terme *application* pour référencer tous les processus d'une même application.

situations suivantes :

– Lors du lancement d'une nouvelle application : l'application est alors sélectionnée pour un placement initial.
– Lors de la terminaison d'une application : toutes les autres applications sont sélectionnées par la politique de transfert, en commencant par les applications les plus prioritaires. Ceci permet de tirer profit des ressources laissées libres par l'application venant de se terminer.
– Lors d'un dépassement de charge sur une machine : la politique de transfert sélectionne alors toutes les applications qui s'exécutent sur la machine surchargée. Une machine est surchargée lorsque sa charge courante dépasse un seuil statique.

Remarquons que lorsqu'une application de forte priorité est démarrée, son lancement peut provoquer la migration (ou dans le pire cas la suspension) d'applications de priorité inférieure. Les applications sont déplacées de manière transparente au programmeur, une migration ne pouvant toutefois se produire que lors d'une barrière de synchronisation.

3.3. *Politique de placement*

Le choix de machines sur lesquelles placer (ou déplacer) les applications dépend dans Stardust à la fois de la charge des machines et des préférences du programmeur des applications. Ce dernier établit un fichier de configuration contenant une liste de *topologies*, trié par préférence décroissante, pour chaque application. Une topologie définit les ressources dont a besoin une application :

– nombre de machines sur lesquels l'application doit être exécutée,
– type d'architecture sur laquelle chaque processus de l'application doit être lancé (ou la constante ANYWHERE, indiquant que le choix de l'architecture est laissé libre au système).

L'utilisateur peut également assigner un processus à une machine fixe en donnant son adresse Internet.

Ainsi, l'utilisateur peut spécifier qu'il désire de préférence utiliser 32 processeurs Paragon pour exécuter son application, et si ce n'est pas possible utiliser 8 PCs, et en dernier ressort utiliser 4 machines de type indifférent.

Quand les processus d'une application sont sélectionnés par la politique de transfert, la liste des topologies est parcourue séquentiellement jusqu'à ce qu'une d'entre elles soit sélectionnée. Une topologie est sélectionnée s'il existe un ensemble de machines qui ne sont pas surchargées pouvant accueillir l'application. Lorsque plusieurs ensembles de machines sont possibles, les machines avec les valeurs de charge les plus faibles sont choisies.

4. Réalisation et performance de Stardust

Stardust est actuellement implanté sur une architecture hétérogène composée d'une machine parallèle Intel Paragon composée de 56 processeurs et d'un ensemble d'une dizaine de machines compatibles PC 90Mhz interconnectées par un réseau ATM à 155Mbits/s (voir figure 1). Les processeurs Paragon (resp. PCs) sont équipés de 16Mo (resp. 32Mo) de mémoire vive. En moyenne, une application s'exécutant sur une machine PC a un temps d'exécution 1.5 fois plus court que la même application s'exécutant sur un processeur de la machine Paragon. Les systèmes d'exploitation utilisés sont le système Paragon/OSF1, à base de micro-noyau Mach, sur la machine Paragon, et le système d'exploitation Chorus/classiX sur les compatibles PC. Pour les deux systèmes, les communications par mémoire virtuelle partagée sont réalisées par construction d'un serveur de pagination externe au noyau. Des détails concernant la réalisation de Stardust peuvent être trouvés dans [2].

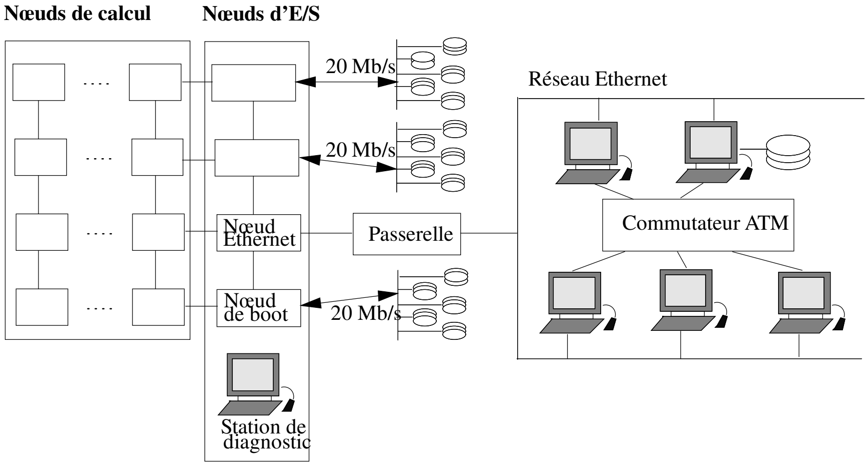

FIG. 1 – *Architecture matérielle utilisée par Stardust*

Le tableau 1 montre les gains de performance qui ont été obtenus sur trois applications grâce au mécanisme de régulation de charge[3]. Pour ces mesures, la seule charge présente sur les machines du système est la charge générée par les trois applications. L'application MatMultHetero4_4 est composée d'une suite de 10 produits de matrices de 256x256 flottants double précision ; une migration depuis 4 nœuds Paragon vers 4 PCs est effectuée après 5 produits. MatMultHomo2_4_8 est le même produit de matrices, pour laquelle 3 produits sont calculés sur 2 PCs, les trois produits suivants sur 4 PCs et les produits restants sur 8 PCs. L'application PovrayHomo2_4_8 est une application de lancer de rayons du domaine public, transformée de manière à utiliser un segment de mémoire virtuelle partagée pour stocker l'image qui est calculée. Les mesures pour cette application ont été obtenues en calculant une image de 1024x728 pixels, chaque pixel étant représenté par une valeur 16 bits. L'exécution comprend deux migrations (une première pour déplacer l'application de 2 à 4 PCs après le calcul de 200 lignes, et la deuxième pour la déplacer de 4 à 8 PCs après le

3. Le driver ATM étant en cours de mise au point, les chiffres figurant dans le tableau ont été obtenus en utilisant un réseau Ethernet à 10Mb/s.

calcul de 400 lignes). Le tableau compare pour chaque application le temps d'exécution sans migration (colonne de gauche) au temps avec migration (colonne de droite). Le chiffre indiqué entre crochets dans la colonne nommée *Avec migration* donne le temps d'exécution théorique de l'application si l'on avait un temps de migration nul. Les migrations sont déclenchées manuellement lors de barrières de synchronisation placées par le programmeur, et tous les processus de l'application sont déplacés.

	Sans migration	**Avec migration**	**Gain** (%)
MatMultHetero4_4	65.2 s (Paragon) 50.2 s (PCs)	61.4 s [56.4 s]	5.8
MatMultHomo2_4_8	69.6 s (2 machines) 36.1 s (4 machines) 20.8 s (8 machines)	46.9 s [40.5 s]	32.6
PovrayHomo2_4_8	3945 s (2 machines) 1992 s (4 machines) 993 s (8 machines)	2074 s [2068 s]	47.4

TAB. 1 – *Performance de l'algorithme de régulation de charge*

Les mesures figurant dans le tableau 1 montrent que des gains de performance importants peuvent être obtenus grâce à l'algorithme de régulation de charge de Stardust. Un gain de 6% est obtenu pour la première application, qui réalise une migration hétérogène. Des gains de 32% et 47% sont observés pour les deux autres applications, pour lesquelles on étend dynamiquement le nombre de machines utilisées pour leur exécution. Ces mesures montrent que le coût de migration de tous les processus d'une application (qui s'échelonne de 2 à 6 secondes pour les trois applications considérées selon leur consommation mémoire), est compensé par le gain en temps d'exécution occasionné par la migration. Pour les mesures présentées dans le tableau 1, ce gain est obtenu par une différence des vitesses d'exécution des processeurs. Des mesures complémentaires sont nécessaires pour montrer que Stardust peut également exploiter des variations dynamiques de la charge des machines.

5. Travaux apparentés et conclusion

De nombreux environnements permettant l'exécution d'applications parallèles audessus de réseaux hétérogènes ont été construits. A la différence de la plupart de ces environnements, Stardust intègre dans un même environnement les trois aspects suivants : (i) support pour l'hétérogénéité ; (ii) support de deux modèles de programmation (envoi de messages et mémoire virtuelle partagée) ; (iii) tolérance aux défaillances et régulation de charge. Tout comme Stardust, les environnements Utopia [8], Condor [6] et Gatostar [5] permettent de réguler la charge d'un ensemble de stations reliées par un réseau. Contrairement à ces environnements, des solutions classiques ont été reprises dans Stardust pour diffuser les informations de charge sur les machines du réseau (seuil de charge fixe). En revanche, Stardust supporte la migration d'applications alors qu'Utopia réalise uniquement un placement initial des processus. Par ailleurs, les applications dans Stardust peuvent être déplacées d'un ensemble de machines vers d'autres machines de type différent (migration hétérogène), ce que ne permettent pas les environnements Condor et Gatostar, implantant exclusivement la migration en milieu homogène. La présence de cette fonctionnalité dans Stardust est réalisée au détriment de la transparence de la migration pour les applications : ces dernières peuvent être déplacées uniquement lors des barrières de synchronisation mentionnées explicitement par le programmeur, et aucune information système (comme les fichiers ouverts) n'est actuellement gérée par Stardust.

Les premières mesures de performances, données dans le paragraphe 4 montrent que l'algorithme de régulation de charge de Stardust permet de réduire de manière significative le temps d'exécution d'applications parallèles, et ceci sans effectuer de modification importante dans le code des programmes. Nos travaux actuels dans Stardust consistent à tester son extensibilité à un nombre de machines plus important.

Références

[1] G. Cabillic, G. Muller, and I. Puaut. The performance of consistent checkpointing in distributed shared memory systems. In *Proc. of the 14th Symposium on Reliable Distributed Systems*, pages 96–105, Bad Neuenahr, Germany, Septembre 1995.

[2] G. Cabillic and I. Puaut. Stardust: an environment for parallel programming on networks of heterogeneous workstations. Technical Report 1006, IRISA, Avril 1996.

[3] C. H. Cap and V. Strumpen. Efficient parallel computing in distributed workstation environments. *Parallel Computing*, 19:1221–1234, 1993.

[4] K. M. Chandy and L. Lamport. Distributed snapshots : Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, Février 1985.

[5] B. Folliot and P. Sens. Gatostar: A fault tolerant load sharing facility for parallel applications. In Springer Verlag, editor, *Proc. First European Dependable Computing Conference*, number 852 in Lecture Notes in Computer Science, pages 581–598, Berlin, Germany, Octobre 1994.

[6] M.J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proc. of 8th International Conference on Distributed Computing Systems*, pages 104–111, San Jose, CA, Janvier 1988.

[7] S. Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14(8):1327–1341, Septembre 1988.

[8] S. Zhou, Zheng X, J. Wang, and P. Delisle. Utopia: A load sharing facility for large, heterogeneous distributed computing systems. *Software Practice and Experience*, 23(12):1305–1336, Décembre 1993.