Calculateurs Parallèles, Vol. 7 - n° 4, pp. 345-366, 1995

Plate-forme de Répartition de Charge et de Tolérance aux Fautes pour Applications Parallèles en Environnement Réparti

Bertil Folliot*, Pierre Sens, Pierre-Guillaume Raverdy

Laboratoire de Méthodologie et d'Architecture des Systèmes Informatiques - MASI
U.R.A. CNRS 818, Institut Blaise Pascal, Université Paris VI
4 place Jussieu, 75252 Paris Cedex 05
E-mail: {folliot,sens,raverdy}@masi.ibp.fr

RÉSUMÉ. *Cet article présente une plate-forme répartie d'exécution pour applications parallèles unifiant un gestionnaire de partage de charge et de tolérance aux fautes. Cette plate-forme, GATOSTAR, repose sur l'unification de deux applications développées au-dessus du système Unix : Gatos et Star. L'apport principal de cette unification est de fournir un support complet et cohérent pour l'exécution performante d'applications parallèles à gros grain et longue durée de vie. Les processus d'applications sont initialement placés sur un ensemble hétérogènes de machines disponibles. Ils sont déplacés au cours de leur exécution en fonction de la charge globale du système ou des fautes de machines. Des mesures de performance en environnement réel montrent l'efficacité et les limites de notre système. Des perspectives du placement dynamique en environnement hybride (composé de machines parallèles connectées à un réseau de stations de travail) sont également présentées.*

ABSTRACT. *This paper presents a distributed fault-tolerant load sharing facility for parallel applications support. This platform, called GatoStar, is based on the integration of two facilities developed on top of Unix: Gatos and Star. The main advantage of this unification is to offer a full and coherent support for efficient execution of coarse grain and long running parallel applications. Application's processes are initially allocated to a set of available heterogeneous hosts. Then, they are migrated during their execution according to the global system load or to hosts failures. Performance measurements in a real environment show the efficiency and the limits of our system. Some Placement perspectives in hybrid environment (composed of parallel machine connected to a workstations network) are also presented.*

* Affiliation : Université Paris VII, UFR d'Informatique

MOTS-CLEFS : systèmes répartis, placement d'applications parallèles, répartition de charge, point de reprise, journalisation de message, placement hybride.

KEY WORDS : distributed systems, parallel applications placement, load sharing, checkpointing, message logging, hybrid placement.

1. Introduction

Les systèmes répartis basés sur des réseaux locaux présentent de nouvelles opportunités pour développer des applications parallèles performantes. Un service d'équilibrage de charge permet aux applications de bénéficier pleinement de la puissance de calcul disponible, et principalement des machines inactives [BER 91ab, BOU 94]. En effet, dans les réseaux de stations de travail, les processeurs sont inutilisés de 33% à 90% du temps [THE 89]. Une étude réalisée dans un des réseaux locaux de l'Université Paris VI (18 stations de travail SUN) sur une période de six mois, montre que les stations sont libres et les processeurs inutilisés, respectivement 69% et 93% du temps.

La répartition des processus d'applications parallèles permet de réduire les temps de réponse et d'améliorer l'utilisation globale des ressources. Nous considérons des applications à longue durée de vie composées d'un ensemble de processus communiquant. Ce modèle couvre un large spectre d'applications scientifiques, comme la factorisation des grands nombres, la conception VLSI ou encore le traitement d'image. Cependant, l'exécution d'applications en environnement réparti est très sensible aux fautes du système. En effet, la panne d'une machine peut entraîner la perte des calculs effectués et/ou une incohérence dans les résultats produits. Ainsi, il est essentiel de fournir, intégré à la répartition de charge, une gestion des fautes.

Cet article présente les services de GatoStar, une plate-forme d'équilibrage de charge et de tolérance aux fautes pour les applications parallèles dans les réseaux de stations de travail. Ce travail repose sur l'unification de deux systèmes indépendants Gatos et Star. Gatos est un gestionnaire de partage de charge qui répartit automatiquement les applications parallèles sur un ensemble hétérogène de sites en fonction d'algorithmes d'allocation multi-critères [FOL 92]. Star est un gestionnaire de tolérance aux fautes qui reprend automatiquement les processus s'exécutant sur des machines défaillantes en utilisant une technique de points de reprise [SEN 94].

Cette unification n'augmente pas seulement l'efficacité des deux systèmes pris indépendamment mais évite également la redondance de nombreux mécanismes. En effet, chacun des systèmes a besoin d'intégrer au moins trois mécanismes communs permettant de maintenir une connaissance globale des processus s'exécutant, de détecter la défaillance de sites, et de transférer des informations concernant l'état de l'ensemble des machines. Les propriétés principales de GatoStar sont :

- **La portabilité** : GatoStar a été conçu comme une couche logicielle, au-dessus d'un système d'exploitation très répandu (Unix). Il n'exige aucun support matériel spécifique ni aucune modification du système d'exploitation.
- **La souplesse d'utilisation** : Les mécanismes de contrôle d'exécution et de traitement des fautes sont transparents à l'utilisateur. Les programmes doivent uniquement être liés à la bibliothèque de programmation de GatoStar. Le programmeur peut cependant adapter les algorithmes d'allocation en fonction des besoins de chaque programme et définir des attractions ou des répulsions entre les composants de son application.
- **Le support d'applications parallèles** : GatoStar inclut des outils permettant de gérer facilement l'exécution de processus à distance (lors de l'allocation ou de la ré-allocation d'un processus), ainsi que des processus interactifs (en redirigeant les flux standards d'entrées/sorties sur la machine de l'utilisateur). Il offre également un espace de nommage global pour supporter les communications et les accès aux fichiers indépendamment de leur localisation.

La section 2 présente les travaux relatifs à la répartition de charge et la tolérance aux fautes dans les systèmes répartis. La section 3 décrit les modèles d'applications et d'environnements utilisés par GatoStar. Nous présentons dans la section 4 la gestion de la répartition de charge, puis succinctement dans la section 5 la gestion des fautes. La section 6 montre les performances de notre prototype en environnement réel. Nous décrivons dans la section 7 des perspectives de placement offertes par les environnements hybrides.

2. Placement et tolérance aux fautes dans les systèmes répartis

Nous présentons dans cette section des travaux effectués sur le placement d'applications parallèles dans les réseaux de stations de travail, puis sur les techniques de tolérance aux fautes reposant sur des points de reprise. Enfin, nous décrivons quatre systèmes intégrant l'équilibrage de charge et la tolérance aux fautes.

2.1. Placement d'applications dans les réseaux de stations de travail

Un grand nombre de travaux ont été entrepris sur le placement d'applications parallèles. Une partie d'entre eux se sont concentrés sur l'allocation statique qui est soit adaptée aux machines multiprocesseurs, sans gérer les interférences entre les utilisateurs ou les applications concurrentes [LO 85, CHA 94]. D'autres travaux sont spécifiques à une certaine classe d'applications [PRI 84, KIM 90], comme par exemple, la répartition de charge d'applications parallèles à structures de données irrégulières [DEN 95].

Un gestionnaire de placement statique bien connu est le système PVM (Parallel Virtual Machine) [GEI 93]. PVM utilise un fichier de configuration géré par l’utilisateur indiquant la liste des machines pouvant servir à l’exécution. PVM est adapté aux environnements dédiés, mais peut induire des surcharges importantes, si plusieurs utilisateurs utilisent les mêmes machines. Il pose également des problèmes d’interférences avec les machines supportant des utilisateurs interactifs.

Le seconde partie des travaux réalisent une répartition dynamique de la charge en fonction de l'état du réseau. Beaucoup d'entre eux ne prennent en compte que la charge des processeurs et ne s’adaptent pas aux comportements des programmes parallèles comme Condor [LIT 88] ou Radio [BER 91a]. Certains travaux, comme le système Utopia [ZHO 93], très proche de Gatos, considèrent les besoins des programmes, mais ne fournissent généralement pas de support pour les applications parallèles (communication, nommage, attraction/répulsion entre les composants de l’application). Utopia est basé sur les expériences de Ferrari et Zhou [FER 87] sur les valeurs des seuils dédiés au placement et sur des mesures de performances effectuées par simulation. Utopia offre une répartition de charge sur des machines ayant des systèmes d’exploitation hétérogènes, et permet également de gérer des multi-réseaux. Comme dans Gatos, il prend en compte les besoins des programmes (temps processeur, mémoire,…) et gère l’hétérogénéité de l’environnement (vitesse des processeurs, capacités mémoires,…). En revanche, Utopia ne gère que des programmes indépendants et ne fournit aucun service pour la communication entre processus.

2.2. Techniques de point de reprise

Les techniques de points de reprise permettent d’assurer le déroulement correct d’applications en environnement réparti non fiable. Régulièrement, les processus d’applications sauvegardent leur état sur support stable. En cas de défaillance, les exécutions sont reprises à partir des derniers points de sauvegarde ou *points de reprise*. Une des difficultés consiste à reprendre l'exécution à partir d'un état global cohérent [CHA 85]. Les mécanismes de reprise décrits dans la littérature peuvent être classés en deux catégories suivant que les points de reprise sont coordonnés ou indépendants.

Dans la première classe d’algorithmes, les processus se synchronisent afin que l'ensemble des points de reprise soit l'image d'un état global cohérent. Si une faute affecte un des processus, tous les processus sont repris à partir de leur dernier point de reprise. Les principaux inconvénients de cette technique sont :

- le surcoût entraîné par les protocoles de synchronisation [KOO 87, MUL 94],
- la charge importante induite par la réalisation d’un point de reprise [ELN 94].

Pour minimiser le coût des synchronisations, plusieurs travaux se sont concentrés sur la manière de réduire le nombre de processus participant à un point de reprise [KOO 87] ou de diminuer le nombre de messages nécessaires à la synchronisation [CRI 91].

Dans la seconde classe d’algorithmes, les processus d'application réalisent leurs points de reprise sans se concerter. Ainsi le coût des points de reprise est réduit.

Cependant, la reprise s'avère plus complexe car il faut trouver un ensemble de points de reprise cohérents. Il peut en résulter des reprises en cascade provoquant un effet domino. Pour éviter ce phénomène, les techniques de points de reprise indépendants sont souvent associées à des méthodes de journalisation de messages [BOR 89, JOH 90, ELN 94].

2.3. Répartition de charge et de tolérance aux fautes

Peu d'expériences ont été reportées sur l'intégration de la tolérance aux fautes à la répartition de charge. Les principaux systèmes sont REM (Remote Execution Manager) [SHO 87], Condor [LIT 88], DAWGS (Distributed Automated Workload sharing System) [CLA 92] et Paralex [BAB 92]. Ces quatre systèmes ont essentiellement été développés comme des gestionnaires de répartition de charge où la tolérance aux fautes n'est qu'une extension limitée.

REM est un gestionnaire pour l'équilibrage de charge qui supporte des applications parallèles. Il fonctionne au-dessus d'Unix sur un réseau de stations de travail. Chaque processus peut créer un fils sur une machine distante et communiquer avec lui. Une réplication active des processus permet aux applications de résister aux défaillances de site (i.e., chaque programme est exécuté N fois). Cependant, la défaillance de la machine de l'utilisateur entraîne l'échec de toute l'application. De plus, le principe de la réplication active entre en conflit avec la politique de répartition de charge, en entraînant en permanence un surcoût de traitement important sur toutes les machines exécutant une copie des programmes répliqués.

Condor et DAWGS s'appuient sur les mêmes principes que GatoStar. Les processus sont initialement alloués sur les machines inutilisées. Un mécanisme de point de reprise permet de déplacer les processus en cours d'exécution afin de mieux répartir la charge ou pour traiter les défaillances de machines. Le principal inconvénient de Condor est qu'il ne gère que des processus indépendants. DAWGS est un gestionnaire plus complet d'équilibrage de charge et de tolérance aux fautes incluant notamment la redirection des entrées/sorties. Les deux principales différences avec GatoStar sont que :

- DAWGS est basé sur des modifications du système d'exploitation, ce qui limite sérieusement la portabilité du système et des applications supportées ;
- le recouvrement de fautes dans DAWGS n'est pas configurable, et un processus affecté par une faute doit attendre pour être repris que la machine défaillante redémarre.

Paralex est un environnement de programmation permettant la conception d'applications parallèles. La politique de placement vise à maximiser les exécutions parallèles et minimiser les communications distantes. Paralex supporte l'occurrence de pannes durant l'exécution des programmes. Il permet aux utilisateurs de spécifier un niveau de tolérance aux fautes pour chaque application. Pour un niveau égal à k, tous les programmes seront répliqués k+1 fois. Cependant, la migration de processus pour équilibrer la charge des machines lors de l'exécution, n'est possible qu'entre les sites contenant une copie. De plus, le degré de tolérance aux fautes initialement spécifié par le programmeur n'est pas maintenu en cas de défaillance d'une des copies.

3. Modèles d'application et d'environnement

Cette section présente les modèles d'application et d'environnement utilisés dans GatoStar.

3.1. Environnement système

GatoStar a été conçu au-dessus du système Unix (SunOS). Il fonctionne sur un ensemble de stations de travail connectées par un réseau local (Ethernet). Dans cet environnement, les fautes de machines sont assez rares. Clark et McMillim [CLA 92] ont estimé à 2,7 jours la fréquence moyenne de pannes sur un réseau local composé de 10 machines.

Dans la plupart de ses versions, Unix n'offre pas de support pour la programmation temps réel. Des temps de réactions très courts, que ce soit au niveau de la répartition ou de la tolérance aux fautes, ne peuvent pas être fournis. Nous privilégions donc les performances en fonctionnement normal au détriment de la rapidité de réaction aux fautes.

Nous considérons que le système est composé de processeurs silencieux sur défaillances (fail-silent) [POW 88]. Une machine fautive s'arrête de fonctionner sans prévenir les autres. Dans cette première version de GatoStar, nous ne traitons pas les partitionnements de réseau.

Les caractéristiques statiques de l'environnement sont définies par l'administrateur du réseau (liste des machines, vitesse des processeurs, capacités mémoire, etc.). La gestion dynamique de l'environnement est assurée par GatoStar (arrêt, faute, redémarrage de machines, ainsi que le calcul de la charge).

3.2. Modèle d'application

Nous avons adopté un modèle classique d'application, composé de processus communicants reliés entre eux par un graphe de précédence. Ce graphe contient également les informations qualitatives suivantes nécessaires pour exécuter l'application, et aidant le gestionnaire de répartition de charge :

- les précédences et les communications entre programmes,
- les fichiers utilisés,
- les attirances et répulsions entre processus,
- les algorithmes d'allocation associés à chaque programme,
- et les différents noms de codes exécutables de chaque programme.

Cette description peut être faite par la concepteur d'applications, soit au moyen du GEL (Gatos External Language), soit en utilisant des fonctions de bibliothèque [FOL 92].

Des exemples de descriptions sont donnés dans les figures 1 et 2. Le programme C peut s'exécuter sur des machines d'architectures Sparc ou DEC. Les algorithmes de placement doivent prendre en compte les besoins en temps processeur de C ainsi que la charge des machines. C doit s'exécuter après la fin des processus A et B, va utiliser le fichier f1, produire des résultats dans le fichier f2 et communiquer avec le programme D. D ne pourra pas être alloué sur la même machine que C (répulsion).

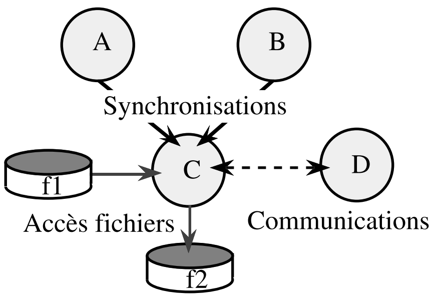

Figure 1. *Exemple de description d'une application*

```
PROG C
       sparc: C.sparc
       dec: C.dec
              ALGO temps, charge
              APRES A, B
              UTILISE f1
              PRODUIT f2
              COMMUNIQUE D DIFF_MACHINE
```

Figure 2. *Exemple de spécification d'un programme en GEL*

Nous nous sommes intéressés aux applications ayant un long temps d'exécution et un parallélisme à gros grain. Cette classe d'applications peut s'exécuter pendant des heures, des jours, voire des semaines. La probabilité de défaillances est alors significative et les besoins de répartition de charge et de tolérance aux fautes deviennent particulièrement importants. Dans la version actuelle, les processus tolérant les fautes doivent être déterministes. Une extension de GatoStar permettant l'exécution de processus non-déterministes est à l'étude.

Les processus de GatoStar sont initialement alloués sur des machines inactives. Au cours de leur exécution, les processus sont déplacés vers des machines compatibles plus appropriées, en fonction des surcharges ou des fautes des machines. Les algorithmes de migration et de placement sont détaillés dans la section suivante.

4. Allocation de processus

Cette section présente succinctement les algorithmes de placement multi-critères et leur extension pour la migration dynamique.

4.1. Placement de processus

Les approches pour la répartition de charge en environnement réparti sont nombreuses [EAG 86, LIT 88, ZHO 93]. Cependant, peu de travaux prennent en compte le placement d'applications parallèles dans un environnement réparti.

4.1.1. Algorithme d'allocation en fonction de la charge

Dans un système homogène, la charge d'une machine est égale au taux d'occupation du processeur (nombre de processus actifs et en attente). Nos algorithmes sont conçus pour un système hétérogène où chaque machine peut avoir des caractéristiques différentes. Ainsi, nous définissons la charge d'une machine comme [BAR 86, ZHO 93] :

- proportionnelle au taux d'occupation du processeur : nombre de processus dans toutes les files d'attente, moins les processus inactifs depuis plus de T secondes[1].
- inversement proportionnelle à la vitesse du processeur (paramètre statique connu à l'avance).

Pour les besoins des algorithmes multi-critères, il est nécessaire de connaître, en plus de la charge, la mémoire disponible et l'activité des disques et du réseau. Par la suite, nous parlerons des informations de charge, qui regroupent tous ces composants.

Pour un programme donné, la sélection d'une machine s'effectue en deux étapes :

- Pour chaque version du programme (compilée sur une architecture différente), la machine compatible ayant la plus faible charge est sélectionnée. Cette selection fournit la liste des meilleures machines en fonction des codes exécutables précisés par l'utlisateur.
- Ensuite, la machine (M) avec la plus faible charge est sélectionnée dans cette liste, et la version correspondante du programme est retenue. Si plusieurs machines ont la plus faible charge, la machine ayant plus grande vitesse processeur est retenue.

Le programme est lancé sur M si la charge de celle-ci est inférieure au **seuil de surcharge**. En effet, dans le cas où toutes les machines sont surchargées, l'exécution distante serait plus coûteuse que l'exécution locale. Le programme est alors lancé sur la machine de départ. Un seuil de surcharge pour les machines ayant

[1] Nous prenons T égale à 20 secondes, comme dans le système UNIX BSD. Un travail futur consistera à évaluer l'influence de cette valeur sur la charge des machines.

un utilisateur connecté permet également d'éviter que ces utilisateurs trouvent leurs machines surchargées par des programmes ne leur appartenant pas.

Le problème principal de cet algorithme (outre la définition de la valeur du seuil de surcharge) consiste à collecter une information globale sur l'état des machines. Un système de répartition de charge transfère généralement ces informations suivant deux grandes approches : **à la demande** ou **à l'initiateur**. Dans l'approche à la demande (ou approche client), les charges de toutes les machines distantes ne sont mises à jour localement que lors du placement. La méthode de mise à jour de la charge à l'initiateur (ou approche serveur) consiste à envoyer périodiquement la charge locale des machines non surchargées à tous les autres sites.

Dans GatoStar, un anneau logique utilisé pour la détection des fautes des machines relie toutes les machines du système (cf. 5.1). Cet anneau véhicule également des informations de charge. Le choix de la mise à jour "à l'initiateur" est donc implicite dans GatoStar. Périodiquement, chaque machine envoie un message de détection à son successeur pour vérifier sa bonne marche. Les informations nécessaires à la répartition de charge sont ajoutées aux messages de détection. Le surcoût induit par ce parasitage est pratiquement nul.

4.1.2. Algorithmes multi-critères

Le but des algorithmes multi-critères est de prendre en compte non seulement l'état des machines (et le respect des utilisateurs interactifs), mais également les différents critères d'allocation donnés dans la description du graphe d'exécution (cf. 3.2). 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.

Vue la complexité de la tâche, le placement multi-critères d'applications parallèles n'est possible que si le programmeur intervient pour diriger le placement par :

- le choix des algorithmes à utiliser pour chaque programme,
- des contraintes de placement au moyen de critères de localisation relative (attraction/répulsion entre programmes ou ressources),
- l'association d'un ensemble de valeurs à chaque programme (temps processeur, mémoire utilisée, volume de communication et d'accès aux fichiers), ou par l'utilisation explicite du mécanisme de suivi d'exécution fourni par GatoStar.

Nous proposons un algorithme pour chaque critère d'allocation. Nos algorithmes fonctionnent chacun de manière indépendante. Plusieurs algorithmes peuvent être combinés en précisant une *liste de critères*. Chaque algorithme fournit alors une liste de machines sélectionnables, et un algorithme de raffinement successif choisi la plus appropriée.

Le concepteur d'application précise lors de la description de son graphe le(s) critère(s) approprié(s) à chaque programme. Il spécifie également les machines qui peuvent être sélectionnées pour le placement (toutes par défaut) et, s'il le désire, fourni des codes de programmes compilés sur différentes machines hétérogènes. Ce dernier point permet d'étendre le placement à toutes les machines pour lesquelles un code exécutable existe. Une description plus complète des algorithmes multi-critères se trouve dans [BOU 92, FOL 92]. Des algorithmes de placement similaires dédiés à la répartition de charge d'applications de grandes tailles à structures de données irrégulières montrent l'intérêt de la prise en compte des besoins de chaque programme [DEN 95].

4.2. Prise en compte des délais de communication et des variations rapides de la charge

Dans un système réparti, il est très difficile de déterminer localement la valeur de l'état global. Ceci est dû principalement aux délais de communication et aux fautes. Dans le cas de la répartition de charge, une connaissance exacte de la charge courante de chaque machine est impossible en raison des :

- délais de communication des informations sur la charge,
- délais entre le moment où un site a pris connaissance des informations sur la charge et le moment où il décide de placer un programme,
- activités des utilisateurs qui n'utilisent pas les services de répartition de charge.

Les variations de charge étant imprévisibles, et les délais précédents étant significatifs, une machine libre au moment de la prise de décision peut devenir surchargée avant le début de l'exécution du processus placé. De plus, si plusieurs allocations ont lieu en parallèle (dans l'intervalle correspondant au délai de diffusion), le même site "libre" peut être choisi pour l'exécution de plusieurs processus créant une surcharge brutale sur ce site (problème de la victime). Cette surcharge ne peut être réduite, après le placement sans un mécanisme de migration (section suivante).

Pour résoudre ces problèmes au moment du placement initial, nous replaçons un programme si la machine cible est devenue lourdement chargée. Ce mécanisme est mis en œuvre en joignant à la requête d'exécution distante la charge estimée de la machine d'accueil. Si la charge courante de cette machine a augmenté significativement, le replacement du programme est effectué.

4.3. Migration de processus

La migration de processus est difficile à réaliser (sauvegarde du contexte / reprise, accès aux fichiers, suivi des communications). Initialement, Gatos n'incorporait pas un tel mécanisme. GatoStar bénéficie directement d'un mécanisme de migration au moyen des techniques de point de reprise/recouvrement développées pour Star (section 4).

Les algorithmes de répartition de charge ont été étendus pour déplacer les processus 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). Le suivi d'application intégré dans la plate-forme permet d'adapter les algorithmes de placement multi-critères en algorithmes de migration multi-critères.

Nous proposons une extension à la migration multicritère prenant en compte la charge et le temps de calcul dans les sections suivantes.

4.3.1. Migration en fonction de la charge des machines

Les trois questions principales qui se posent pour la migration dynamique de processus sont : quand déplacer un processus, lequel sélectionner, et où le déplacer.

La réponse que nous proposons à la première et troisième question repose sur l'utilisation de deux seuils opposés : un **seuil de migration** et un **seuil de réception**. Sur chaque site où s'exécute au moins un processus GatoStar, un serveur local évalue périodiquement deux conditions :

- une machine puissante est en dessous du seuil de réception ou,
- la machine locale est au-dessus du seuil de migration et au moins une machine est en dessous du seuil de réception.

La première condition permet de bénéficier des machines puissantes dès qu'elles sont libérées, et la deuxième fait décroître la charge de la machine locale. Si l'une des deux conditions est vérifiée, le serveur entre dans la phase de sélection de processus. En cas de surcharge, c'est le serveur de la machine surchargée qui gère diirectement cette phase. En cas de sous-charge, le site sous-chargé propose ses services aux autres machines, qui peuvent entrer dans la phase de sélection si au moins un processus GatoStar s'exécute et si la machine cible a une architecture compatible avec un facteur de vitesse sensiblement supérieur à la machine source. Ce facteur de vitesse permet de profiter des machines puissantes lorsqu'elles se libèrent, et garantit l'utilité de la migration.

4.3.2. Sélection des processus candidats à la migration

Tous les processus GatoStar ayant été exécutés localement pendant un temps minimum sont candidats à la migration. Ce critère empêche un processus de passer son temps à être déplacé de machine en machine. Il empêche également les processus à courte durée de vie d'être candidats à la migration. Si plus d'un processus peuvent être déplacés deux cas se présentent. Si aucune information sur le temps d'exécution des programmes n'a été fournis par le concepteur de l'application, le plus vieux processus est sélectionné (temps d'exécution cumulé depuis son démarrage). Sinon, les informations du graphe d'exécution sont utilisées. Dans ce cas, le processus sélectionné est celui qui demande le plus de processeur et qui est supposé avoir le plus long temps d'exécution restant.

Le problème de la victime (plusieurs machines demandent à déplacer des processus vers le même site) est évité au moyen d'un mécanisme de réservation. En effet, la migration d'un processus est un mécanisme lourd, et la méthode de redirection utilisée pour le placement n'est pas utilisable. Ainsi, avant de déplacer un processus, un message est envoyé à la machine de destination pour confirmer que celle-ci répond toujours aux conditions initiales et pour la verrouiller contre d'éventuelles migrations ou placements concurrents.

GatoStar initie également une phase de migration lorsqu'un utilisateur interactif se connecte à sa machine, ou reprend son travail. Les processus GatoStar locaux sont alors suspendus, puis des machines cibles sont sélectionnées. Cette sélection se fait de la même manière que lorsqu'une machine dépasse le seuil de migration. Ce mécanisme permet de minimiser les interférences avec les utilisateurs interactifs.

5. Gestion des défaillances

Cette section présente les services de tolérance aux fautes intégrés dans GatoStar. Nous décrivons, tout d'abord, le mécanisme de détection des défaillances puis la technique de reprise fondée sur la journalisation des messages. Enfin, le support stable est brièvement présenté.

5.1. Détection des défaillances

On distingue deux techniques de détection :

- la détection **passive** où la machine défaillante n'est détectée que lorsque l'utilisateur y accède,
- la détection par **surveillance** où les sites sont scrutés périodiquement.

La détection passive entraîne un surcoût minimum, cependant, le temps de recouvrement des défaillances peut être très important car il dépend du trafic sur le réseau. La surveillance pallie ce problème en instaurant un trafic de contrôle supplémentaire.

Dans GatoStar, nous utilisons ces deux techniques. Le trafic normal est utilisé pour recueillir des informations sur les sites. Lorsque ce trafic est insuffisant, des messages de détection sont générés. Pour limiter le coût du protocole de surveillance, les sites sont organisés en anneau. Chaque machine surveille alors uniquement son voisin de droite. Ce mécanisme est très simple et exige peu de messages. De plus, les messages de détection sont également utilisés pour véhiculer sans coût supplémentaire des informations de charge. En revanche, cette technique nécessite des protocoles d'insertion de nouveaux sites et reconfiguration en cas de défaillances [BOR 89].

5.2. Reprise sur défaillance

En cas de défaillances, les processus situés sur les sites fautifs sont réexécutés sur des sites valides. Notre plate-forme offre aux applications une technique de points de reprise indépendants avec stockage des messages sur support stable afin d'éviter l'effet domino lors de la reprise [SEN 94]. Ces messages sont alors "rejoués" localement lors de la réexécution du processus jusqu'à l'instant de la panne. Ainsi, un processus repris se réexécute indépendamment des autres processus à partir des données sauvegardées. Ce principe est mis en œuvre par des techniques de journalisation de messages et de détection des réémissions :

- Chaque processus sauvegarde dans un journal l'ensemble des messages reçus depuis le dernier point de reprise. Ainsi, en cas de défaillance de son site d'exécution, le processus repris consomme directement les messages du journal.
- Un processus repris va réémettre ses messages. De façon à éviter de perturber l'exécution des processus valides, un mécanisme de détection des messages réémis, basé sur des estampilles, permet de les supprimer.

Cette approche présente les avantages suivants :

- **Faible coût des points de reprise** : les processus créent leurs points de reprise sans synchronisation.
- **Reprise rapide** : la reprise est limitée aux seuls processus fautifs. De plus, lors des réexécutions les communications sont simulées localement diminuant ainsi le trafic sur le réseau.
- **Un seul point de reprise par processus** : c'est la conséquence directe de l'absence d'effet domino. De cette manière, l'espace de stockage associé à chaque processus est réduit.

L'inconvénient majeur résulte de la sauvegarde de tous les messages sur support stable qui entraîne un surcoût important des communications et peut nécessiter des ressources de stockage importantes. Par exemple, si le support stable est implanté sous forme de fichiers dupliqués, la journalisation ralentit l'envoi d'un message de moitié (voir section 6.2).

5.3. Support stable

Le support stable est la clé de voûte de la gestion des fautes. Il est utilisé pour stocker les points de reprise et les journaux de messages. Les données sur support stable sont stockées dans des fichiers répliqués sur plusieurs sites. Le degré de réplication est spécifié par l'utilisateur au lancement de son application. Pour un degré de réplication de N, N-1 fautes simultanées sont tolérées. La cohérence, lors des mises à jour des réplicas, est garantie par un protocole de diffusion fiable basé sur [CHA 84]. De plus, en cas de fautes, le degré de réplication initial est maintenu dans la mesure des ressources restantes (machines et disques).

6. Performances

Nous présentons, dans cette section, quelques mesures de performance de l'algorithme d'allocation multi-critères et du gestionnaire de tolérance aux fautes. Ces mesures ont été effectuées en environnement réel dans notre laboratoire (environnement de recherche et universitaire). Le système est composé de stations de travail Sun (SunOS, Sparc Station 1, 16Mo de mémoire centrale) reliées par un réseau local (Ethernet à 10Mo/s).

L'évaluation présentée repose sur une implémentation multi-serveurs [FOL 94]. Sur chaque machine, un serveur de placement implémente les algorithmes d'allocation, un serveur de reprise est responsable de la gestion des fautes, un serveur de fichier implémente le support stable et un serveur d'anneau est responsable de l'acquisition des informations de charge et de la détection des fautes.

6.1. Allocation de processus

La surcharge moyenne pour placer dynamiquement un processus est de moins de 100 millisecondes. Ce temps est de 5 secondes dans le système REM [SHO 87], 500 millisecondes dans le système Radio [BER 91a], et 12 secondes dans DAWGS [CLA 92]. Ces temps importants sont principalement dus aux communications entre les serveurs, et au temps d'attente avant de confirmer l'ordre de placement (égale à 8 secondes dans DAWGS). Dans GatoStar, aucune attente n'est nécessaire et aucun message n'est échangé lors de l'allocation.

Notre algorithme d'allocation classique ne prend en compte que la charge des machines. Il divise en moyenne par deux le temps de réponse d'une application parallèle composée de quatre programmes non communicants dans un réseau de cinq sites. Le temps de réponse minimum est proche de l'optimal (gain de 3.9 sur quatre machines).

L'algorithme d'allocation multi-critères prenant en compte le temps prévu d'exécution diminue le temps de réponse de 14 à 20% par rapport à l'algorithme classique. La figure 3 indique le gain obtenu pour un système composé de trois sites en fonction de la complexité des applications. Par exemple, l'application A (2.10M, 1.50M) présente 2 programmes effectuant 10 millions d'itérations en parallèle avec 1 programme effectuant 50 millions d'itérations (d'un calcul identique).

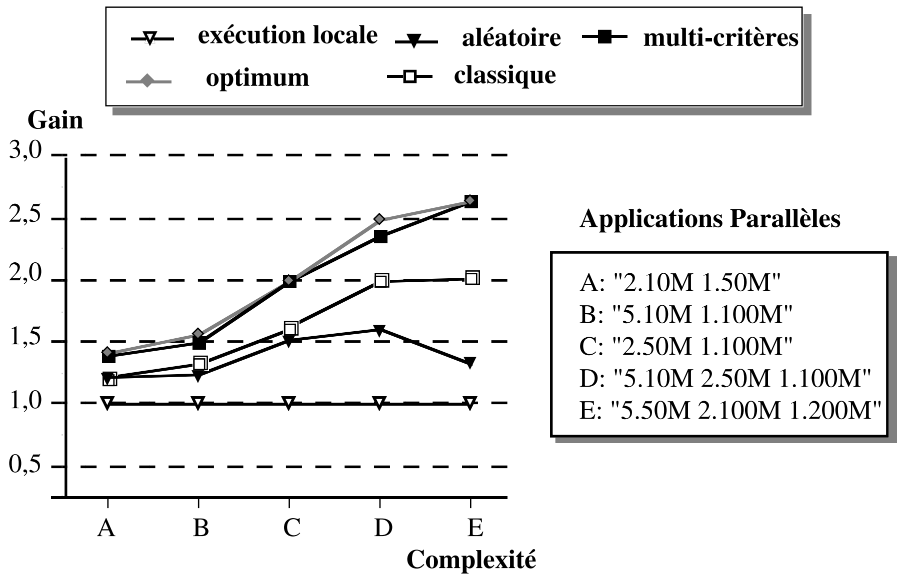

Figure 3: Gain en fonction de la complexité des applications

6.2. Gestion des fautes

Nous avons également évalué le surcoût de la gestion des fautes pour trois applications parallèles typiques [SEN 95]. La première, gauss, est une élimination gaussienne parallèle. Cette application exige beaucoup de mémoire (2,6 Mo par processus) et d'importantes communications (2,7 Mo émis par chaque processus). La seconde application, matmul, est une multiplication de matrice parallèle. matmul requiert moins de mémoire (1,7 Mo par processus) et pratiquement aucune communication outre les synchronisations initiales et finales. La dernière application testée, fft, est une transformée de Fourier rapide. Les processus sont de taille moyenne (1,2 Mo) et n'exigent pratiquement pas de communications. Pour ces trois applications, le traitement est réparti sur quatre sites. La table 1 montre le surcoût obtenu pour une configuration du support stable avec un degré de réplication de deux et une fréquence de points de reprise de 2 minutes.

Applications	Surcoût en %
gauss	32,85 %
matmul	3,50 %
fft	1,50 %

Table 1. *Surcoût de la gestion des fautes*

Le surcoût est très faible pour les applications matmul et fft tandis que pour gauss le surcoût est assez important. En fait, pour cette dernière, la journalisation représente environ la moitié du surcoût total. Ces mesures montrent l'efficacité de la technique de point de reprise indépendant particulièrement pour des applications dont les processus échangent peu d'informations.

7. Extension du placement dynamique en environnement hybride

Les solutions présentées dans GatoStar sont limitées au cadre d'un réseau local de stations de travail. L'interconnexion de machines parallèles dans des réseaux de stations de travail offre de nouvelles opportunités pour la répartition d'applications parallèles comportant plusieurs types de parallélisme (calcul hétérogène à très hautes performances) [KHO 93]. Le placement dans de telles architectures, appelées environnement hybride, doit intégrer plusieurs modèles de calculs parallèles de manière à profiter des avantages combinés des différentes granularités de parallélisme : petit, moyen ou gros grain.

Dans cette section nous présentons tout d'abord la problématique de placement dans les environnements hybrides, puis nous proposons un modèle pour la répartition de charge entre systèmes réparti et parallèle.

7.1. Perspectives du placement hybride

Les réseaux de stations de travail (Sun, HP, DEC...) connaissent un très fort développement. De plus, les machines parallèles (Paragon, CM5, T3D...) sont de plus en plus souvent intégrés à ce type de réseau. Ces architectures hybrides sont caractérisés par un grand nombre de noeuds, une grande diversité des types de ressources et surtout par le support simultané de différents grains de parallélisme.

L'exécution d'une application sur un réseau de machines parallèles permet de choisir dynamique une machine parallèle en fonction de son utilisation. La programmation spécifiques sur ces architectures, appelée program-mation hybride, permet d'exécuter les différentes parties de l'application sur l'architecture matérielle la plus adaptée (vectorielle, multiprocesseurs, stations de travail) et ainsi de réduire les principaux goulots d'étranglement liés à une architecture [BRE 94, EDJ 94].

Les systèmes répartis et les machines parallèles (plus particulièrement les machines massivement parallèles) n'offrent pas les mêmes possibilités ni les mêmes contraintes pour la répartition de charge.

D'un côté, les réseaux de stations de travail sont fortement hétérogènes en terme d'architecture, de puissance de calcul, de ressources (capacité mémoire, disque, matériel spécialisé) et sont plus adaptés à des applications parallèles à moyen ou à gros grain. Les principales limites de ces réseaux se situent dans les communications entre processus (haute latence et faible débit) et dans le faible nombre de stations participant à la répartition de charge (en général limité au réseau local).

D'un autre côté, les machines parallèles ont des temps de communication très faibles, une architecture homogène et peuvent avoir un nombre de noeuds important. Ces machines sont adaptées à un calcul parallèle à fin ou moyen grain, où les processus s'échangent une quantité importante de données.

Ces caractéristiques opposées ont amené, dans chacun des domaines, à des solutions différentes pour la réalisation d'environnements pour la programmation parallèle, intégrant l'équilibrage de charge et la tolérance aux fautes. Quelques travaux ont considéré le problème du placement sur des machines parallèles homogènes répartis dans un réseau local (machines DEC Firefly [CHA 89] ou, machines à base de transputer [GUY 93]).

Les principaux problèmes à résoudre pour un placement hybride hétérogène sont les suivants :

- Définir un mécanisme de communication pour applications parallèles prenant en compte les problèmes spécifiques aux réseaux hybrides (nommage, adressage, routage à plusieurs niveaux).
- Définir les mécanismes de description des applications hybrides et de l'environnement hétérogène.
- Définir les algorithmes de placement appropriés à ces nouveaux modèles d'applications et d'environnement permettant de considérer les différents grains de parallélisme dans une même application.

7.2. Modèle coopératif de placement hybride

Un placement uniforme en environnement hybride n'est pas possible en raisons des différences dans les mécanismes suivants :

- la granularité de l'information sur chaque noeud de l'environnement,
- la politique d'échange de l'information entre les noeuds,
- l'algorithme d'allocation qui dépend fortement de la granularité de l'information, de la politique d'échange, mais aussi de la topologie du réseau.

Nous proposons un modèle coopératif de placement, préservant les spécificités de chaque architecture pour la répartition de charge tout en permettant à chacun de ces environnements de bénéficier de la puissance de l'autre.

Notre modèle est basé sur les systèmes de répartition de charge développés au laboratoire MASI, l'un adapté aux systèmes répartis, GatoStar, l'autre adapté aux machines massivement parallèles, les Territoires d'Exécution [RAV 95].

Dans le modèle des territoires, le domaine de calcul (domaine composé de tous les noeuds de la machine parallèle) est découpé en territoires disjoints. Un territoire correspond à un ensemble de processeurs réservés pour l'exécution des processus d'une même application. L'échange d'informations est réalisé uniquement à l'intérieur d'un territoire, ce qui réduit la quantité d'informations échangée et la complexité des décisions liées au partage de charge.

Le placement hiérarchique en environnement hybride que nous proposons se décompose en deux parties :

1) Une machine parallèle est sélectionnée pour chaque partie de l'application dédiée à une architecture spécifique. Cette sélection est effectuée par le système de placement du réseau local (GatoStar) en fonction des informations sur la charge des machines parallèles (fournies par les gestionnaires des domaines de calcul des Territoires d'Exécutions).
2) D'une manière autonome les différents gestionnaires des domaines de calcul (pour les machines parallèles) et GatoStar (pour les stations de travail) sélectionnent les processeurs pour l'allocation des parties de l'application qu'ils doivent exécuter.

Ce modèle coopératif permet de bénéficier des différents grains de parallélisme qui émergent dans les réseaux locaux. Il ouvre également la voie vers un découpage hybride de calculs scientifiques ou d'application parallèles existantes.

8. Conclusion

Nous avons présenté la conception d'une plate-forme de répartition de charge et de tolérance aux fautes, GatoStar, dédiée aux applications à gros grain, ayant peu de communications et s'exécutant sur un réseau de stations de travail. Cette plate-forme permet de bénéficier de l'ensemble des ressources du réseau en répartissant les processus des applications sur les sites les plus appropriés. La politique de placement des processus prend en compte non seulement l'état du système lors du lancement des processus mais aussi les besoins des applications. Un mécanisme de point de reprise permet de reprendre l'exécution de processus fautifs suite à la défaillance d'un site. La cohérence du système est maintenue par un mécanisme de journalisation de message.

La réalisation de la plate-forme GatoStar repose sur les deux systèmes Gatos et Star. Le même anneau virtuel de communication est utilisé pour transmettre la connaissance de l'état du système et pour détecter les fautes des machines. Un premier prototype de GatoStar a été réalisé au dessus de SunOs. Nous sommes en train de le porter au-dessus de Linux (POSIX) et DEC/Ultrix. GatoStar présente les avantages suivants :

- Il fournit aux programmeurs une base pour construire des applications parallèles tolérantes aux fautes avec un placement multi-critères adapté aux besoins individuels des programmes.
- Il réduit la surcharge liée au fonctionnement indépendant des deux services (Gatos et Star), en diminuant le nombre de messages échangés et en supprimant les fonctionnalités redondantes (gestion des processus, maintien d'un état global).
- Il permet un traitement efficace des fautes en choisissant le site de reprise d'un processus fautif en fonction de la charge du système.

Des mesures de performance montrent l'intérêt de notre plate-forme pour l'exécution d'applications parallèles. Dans un environnement réel la moyenne du gain obtenu pour trois programmes parallèles s'exécutant sur trois machines et de 2,2. La prise en compte du comportement des programmes améliore fortement ce gain (jusqu'à 2,8). De plus, pour pallier aux changements impromptues de charge, une extension des algorithmes de placement multi-critères a été proposée. Cette dernière se base sur les possibilités de migration de GatoStar pour déplacer les processus soit en cas de surcharge des machines (pour bénéficier des autres machines inactives), soit en cas de sous-charge (pour profiter de l'hétérogénéité de la puissance de calcul).

En perspective, nous avons proposé une extension du placement afin de gérer des réseaux hybrides composés de machines multiprocesseurs hétérogènes connectées à un réseau de stations de travail. Cette extension permettra de profiter de la puissance de calcul cumulée des stations de travail et des machines parallèles, et offrira de nouvelles opportunités pour la conception d'applications hybrides, en adoptant le meilleur modèle de programmation parallèle pour chaque partie d'une application.

Bibliographie

[ALO 88] R. Alonso et L.L. Cova. Sharing Jobs Among Independently Owned Processors. In Proc. of the 8th International Conference on Distributed Computing Systems, San Jose, Californie, pp. 365-372, février 1988.

[BAB 92] O. Babaoglu, L. Alvisi, A. Amoroso, et R. Davoli. Paralex: An Environment for Parallel Programming in Distributed Systems. *Proc. of International Conference on Supercomputing*, Washington D.C., pp. 178-187, juillet 1992.

[BAR 86] A. Barak et O.G. Paradise. MOS - A Load-balancing Unix. In *Proc. of the Usenix Technical Conference - Summer 1986*, pp. 414-418, 1986.

[BER 91a] G. Bernard et M. Simatic. A Decentralized and Efficient Algorithm for Networks of Workstations. In *Proc. of the European Conference for Open Systems Spring '91*, Tromsø, Norvège, pp. 139-148, mai 1991.

[BER 91b] G. Bernard, D. Stève, et M. Simatic. Placement, Migration de Processus dans les Systèmes Répartis Faiblement Couplés. *Technique et Sciences Informatiques*, 10(5):375-392, décembre 1991.

[BOR 89] A. Borg, W. Blau, W. Craetsch, F. Herrmann, et W. Oberle. Fault Tolerance under Unix. *ACM Transactions on Computer Systems,* 7(1):1-24, février 1989.

[BOU 94] P. Bouvry, D. Trystram, et J.M. Geib. Répartition de charge. *Algorithmes parallèles : analyse et conception*, G. Authié, A. Ferreira, J.L. Roch, G. Villard, J. Roman, C. Roucairol, B. Virot Eds., Editions Hermès, pp. 203-235, 1994.

[BOU 92] R. Boutaba et B. Folliot. Load Balancing in Local Area Networks. In *Proc. of the Networks'92 International Conference on Computer Networks, Architecture and Applications*, Trivandrum, India, pp. 73-89, octobre 1992.

[BRE 94] C. Brezinski et M.R. Zaglia. A Hybrid Procedure for Solving Linear Systems. *Numer. Math.*, 67, pp. 1-19, 1994.

[CHA 85] K.M. Chandy et L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, 3(1):63-75, février 1985.

[CHA 84] J. Chang et N. F. Maxenchuk. Reliable Broadcast Protocols. *ACM Transaction on Computer Systems*, 2(3):251-273, août 1984.

[CHA 94] C-S. Chang, R. Nelson, et D. Yao. Optimal Task Scheduling on Distributed Parallel Processors. *Performance Evaluation* Vol 20, pp. 207-221, 1994.

[CHA 89] J. Chase, F. Amador, E. Lazowska, H. Levy, R. Littlefield. The Amber System: Parallel Programming on a Network of Multiprocessors. In *Proc. of 12th ACM Symp. on Operating System Principles*, pp. 147-158, septembre 1989.

[CLA 92] H. Clark et B. McMillin. DAWGS - A Distributed Compute Server Utilizing Idle Workstations. *Journal of Parallel and Distributed Computing*, 14:175-186, février 1992.

[CRI 91] F. Cristian et F. Jahanian. A Timestamp-based Checkpointing Protocol for Long-lived Distributed Computations. *Proc. of 10th Symposium on Reliable Distributed Systems*, pp. 12-20, septembre 1991.

[DEN 95] Y. Denneulin, J.M. Geib, C. Gransart, F. Hémery, J.F. Méhaut, J.F. Roos, E.G. Talbi, M. Christaller, T. Gautier, J.L. Roch, et A. Vermeerbergen. Structure de Données Irrégulières et Répartition de Charge. Actes de l'Ecole d'Hiver, Conception et Analyse des Algorithmes Parallèles, Cauterets, France, pp. 211-247, février 1995.

[EAG 86] D. L. Eager, E. D. Lazoska, et J. Zahorjan. Adaptative Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering*, SE12(5):662-675, mai 1986.

[EDJ 94] G. Edjali, N. Emad, S. Petiton. Hybrid Methods on Network of Heterogeneous Parallel Computers. In *Proc. of the 14th IMACS World Congress.* Atlanta, juillet 1994.

[ELN 94] E.N. Elnozahy et W. Zwaenepoel. On the Use and Implementation of Message Logging. *Proc. of the 24th International Symposium on Fault-Tolerant Computing Systems* (FTCS 24), pp. 298-307, Austin, Texas, juin 1994.

[FER 87] D. Ferrari et S. Zhou. An Empirical Investigation of Load Indices for Load Balancing Applications. *Performances '87*, Bruxelles, Belgique, pp. 515-528, décembre 1987.

[FOL 92] B. Folliot. Méthodes et Outils de Partage de Charge pour la Conception et la Mise en Œuvre d'Applications dans les Systèmes Répartis Hétérogènes. *Thèse de Doctorat de l'Université Paris 6,* Research Report N°93-27, Institut Blaise Pascal, décembre 92.

[FOL 94] B. Folliot, P. Sens. GatoStar: A Fault Tolerant Load Sharing Facility for Parallel Applications. In Proc. *First European Dependable Computing Conference*, IFIP, IEEE, Berlin, Germany, octobre 1994. *Lecture Notes in Computer Science 852*, K. Echtle, D. Hammer, D. Powell Eds., Springer-Verlag, pp. 581-598, 1994.

[GEI 93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, et V. Sunderam. PVM 3 User's Guide and Reference Manual. Oak Ridge National Laboratory, *Technical Report* 12187, mai 1993.

[GUY 93] H. Guyennet et F. Spies. A Cooperative Algorithm for Load Balancing in Interconnected Transputer Network. In *Proc. of International Conference on Decentralized and Distributed Systems*, Palma de Mallorca, pp. 185-195, septembre 1993.

[JOH 90] D.B. Johnson et W. Zwaenepoel. Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing. *Journal of Algorithms*, 11(3):462-491, septembre 1990.

[KIM 90] C. Kim et H. Kameda. Optimal Static Load Balancing of Multi-class Jobs in a Distributed Computer System. In *Proc. of the 10th International Conference on Distributed Computing Systems*, Paris, France, mai 1990.

[KHO 93] A. Khokhar, V. Prasanna, M. Shaaban, et C.L. Wang. Heterogeneous Computing: Challenges and Opportunities. *COMPUTER*, pp. 18-27, juin 1993.

[KOO 87] R. Koo et S. Toueg. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, SE-13(1):23-21, janvier 1987.

[LIT 88] M. J. Litzkow, M. Livny, et M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proc. of the 8th International Conference on Distributed Computing Systems*, San José, Californie, pp. 104-111, janvier 1988.

[LO 85] V.M. Lo. Task Assignement to Minimize Completion Time. In *Proc. of the 5th International Conference on Distributed Computing Systems*, pp. 239-336, 1985.

[MUL 94] G. Muller, M. Hue, et N. Peyrouze. Performance of Consistent Checkpointing in a Modular Operating System: results of the FTM Experiment. *First European Dependable Computing Conference*, IFIP, IEEE, Berlin, Germany, octobre 1994. *Lecture Notes in Computer Science 852*, K. Echtle, D. Hammer, D. Powell Eds., Springer-Verlag, pp. 491-508, 1994.

[POW 88] D. Powell, G. Bonn, D. Seaton, P. Verissimo, et F. Waeselynck. The Delta-4 Approach to Dependability in Open Distributed Computing Systems. In *Proc; of the 18th International Symposium on Fault-Tolerant Computing Systems*, Tokyo, Japan, pp. 246-251, 1988.

[PRI 84] C.C. Price et S. Krishnaprasad. Software Allocation Models for Distributed Computing Systems. In *Proc. of the 4th International Conference on Distributed Computing Systems*, San Fransisco, pp. 40-48, mai 1984.

[RAV 95] P.-G. Raverdy, B. Folliot. Presentation of the Execution Territory: a Two Levels Load Balancing Mechanisms. A paraître dans les actes *European Research Seminar on Advances in Distributed Systems*, l'Alpe d'Huez, avril 1995

[SEN 94] P. Sens. Conception et Mise en œuvre d'une Plate-forme Logicielle de Tolérance aux Fautes pour le Support d'Applications Réparties. *Thèse de Doctorat de l'Université Paris 6*, décembre 1994.

[SEN 95] P. Sens. The Performance of Independent Checkpointing in Distributed Systems. In *Proc. of the 28th Hawaii International Conference on System Sciences* - Software Technology Track, pp. 525-535, janvier 1995.

[SHO 87] G. C. Shoja, G. Clarke, et T. Taylor. REM: A Distributed Facility For Utilizing Idle Processing Power of Workstations. In *Proc. of the IFIP Conference on Distributed Processing*, Amsterdam, octobre 1987.

[THE 89] M. M. Theimer et K. A. Lantz. Finding Idle Machines in a Workstation-based Distributed System. *IEEE Transactions on Software Engineering*, 15(11):1444-1458, novembre 1989.

[ZHO 93] S. Zhou, X. Zheng, J. Wang, et P. Delisle. Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software - Practice and Experience*, Vol 23 (12), pp. 1305-1336, décembre 1993.