Année : **2011** Numéro d'ordre : **40563**

Habilitation à Diriger des Recherches
de l'Université de Lille I

Discipline : Mathématique

Passage à l'échelle d'applications java distribuées auto-adaptatives

Par : **Richard OLEJNIK**

Sous la direction de Bernard TOURSEL, Professeur des Universités

Membres du jury :

- Rapporteur : Beniamino DI MARTINO, Professeur, Second University of Naples
- Rapporteur : Thierry PRIOL, Directeur de Recherche, INRIA
- Rapporteur : Marek TUDRUJ, Professeur, Académie des Sciences de Pologne
- Examinateur : Pierre BOULET, Professeur des Universités, Lille I
- Examinateur : Christophe CERIN, Professeur des Universités, Paris 13

Date de soutenance : 26 Septembre 2011

Remerciements

Le travail présenté dans le manuscrit a été effectué au sein de l'équipe PALOMA du Laboratoire d'Informatique Fondamentale de Lille, dont je voudrais remercier les directeurs successifs (les Professeurs Max Dauchet, Bernard Toursel, Jean Marc Geib et Sophie Tison), pour m'avoir accordé toute la liberté nécessaire pour la réalisation de mes projets.

Je tiens particulièrement à remercier le Professeur Bernard Toursel pour m'avoir donné l'opportunité d'intégrer l'équipe « PArallélisme LOgiciel et MAtériel » (PALOMA). Je le tiens en très haute estime et travailler avec lui fut très enrichissant et formateur pour moi

Mes remerciements vont ensuite aux membres du jury pour l'honneur qu'ils me font en y participant. Je remercie vivement Monsieur Thierry Priol, Directeur de Recherche INRIA et le Professeur Christophe Cerin d'avoir accepté de passer du temps pour étudier mon travail. Je remercie les Professeurs Marek Tudruj et Beniamino Di Martino ainsi que tous les membres de leurs équipes, avec lesquels j'ai le très grand plaisir de travailler. Je remercie enfin le Professeur Pierre Boulet, avec qui s'amorcent de nouvelles collaborations qui m'ouvrent des perspectives de recherches intéressantes.

Je remercie également tous les membres de l'équipe PALOMA qui ont permis l'aboutissement de ces dix années de recherches. Je remercie notamment tous les étudiants qui ont participé de près à ces travaux et avec lesquels des liens très chaleureux se sont noués : Amer, Violeta, Iyad, Valérie, Mahmoud, Guilhem, Guillem, Xiobey, sans oublier les nombreux étudiants de DESS, maitrises, masters et de l'Ecole d'Ingénieurs Polytech'Lille.

Mes remerciements s'adressent aussi à mes collègues chercheurs, ITA, et enseignants. L'ambiance qui a toujours régné au laboratoire, est vraiment propice à la réalisation d'un travail de qualité.

Merci enfin à Dominique mon épouse et mes enfants, Cécile et Sylvain, ainsi qu'à tous mes proches qui m'apportent quotidiennement leur soutien.

Table des matières

Introduction

Les applications ont sans cesse besoin de plus de puissance de calcul. Bioinformatique, prédictions météorologiques, neuroimagerie, simulations militaires … Autant de recherches rendues possibles grâce à la phénoménale augmentation de puissance des ordinateurs et au développement du calcul distribué, qui permet de mutualiser la puissance de calcul des réseaux informatiques. L'exemple type en est sans doute le séquençage du génome humain et la comparaison des séquences d'ADN ou de protéines par des méthodes de bioinformatique. L'entreprise n'aurait pas été possible sans les puissantes machines connectées en réseau qui ont assemblé virtuellement les fragments du génome humain, séquencés les uns après les autres dans des dizaines de laboratoires. Mais, la mise en œuvre de tels desseins représente aussi un problème technique. Comment trouver la puissance de calcul nécessaire?

Le calcul distribué relève ce formidable challenge que constitue l'exécution de très grandes applications parallèles sur réseaux d'ordinateurs et permet de lancer des projets autrefois inimaginables. Du point de vue de la programmation, il se caractérise par l'absence d'une mémoire commune, le coût des communications à travers le réseau (notamment en termes de latence) et par l'absence d'un système commun disponible directement. Les applications distribuées comportent un certain nombre de tâches coopérantes qui exploitent les ressources d'une plate-forme d'ordinateurs faiblement couplés. Une application peut simplement être distribuée de par sa nature, pour gagner en performance d'exécution ou pour utiliser des ressources qui n'existent pas en local.

Plusieurs architectures spécifiques peuvent être utilisées comme cible**s**, dans le cadre du calcul distribué : les systèmes multiprocesseurs et/ou multicoeurs, le calcul sur grappe (cluster computing) et le calcul sur grille (grid computing). Les deux derniers concepts sont apparus successivement, durant ces dernières décennies. Tous deux mettent en réseau des ordinateurs afin de les faire travailler sur un même projet. Ils offrent une couche virtuelle entre l'application (processus) et la plate-forme physique. Cette virtualisation poussée à l'extrême dans la mesure où tout le fonctionnement se base sur des machines virtuelles, a conduit à l'apparition du calcul en nuage (cloud computing). Ces systèmes sont très en vogue actuellement, mais nous en reparlerons plus tard. Pour revenir aux grappes et aux grilles, leur différence repose d'abord sur leur échelle, c'est-à-dire sur le nombre d'ordinateurs (ou de nœuds) impliqués dans les calculs. Une seconde différence**,** non négligeable**,** réside dans le fait que pour la grappe, l'utilisateur possède toutes ses ressources de calcul et les connaît dès le départ. Ce qui n'est pas du tout le cas de l'utilisateur de la grille qui peut accéder à plus de ressources qu'il ne dispose en propre. Au final, la grappe comme la grille, additionnent toutes deux, à moindre coût, les puissances de calcul ou de stockage des nœuds de la plate-forme. Elles bénéficient ainsi de la puissance potentielle de calcul, qui s'accroît régulièrement, de manière spectaculaire. Ceci est particulièrement vrai, si l'on veut bien considérer la puissance de calcul fournie**,** non seulement par les ordinateurs que nous utilisons chaque jour,

mais aussi par les moyens mis à disposition par les centres de calcul privés ou publics, et le développement des réseaux à très haut débit.

Le calcul sur grappe abordé dans les deux premiers chapitres, mais plus encore, celui sur grille de calcul développé ensuite, tente de combler le fossé entre ces deux mondes. Ces approches partagent une démarche commune. Elles rassemblent au sein d'une seule et même entité virtuelle, toute la puissance de machines distribuées géographiquement sur différentes plates-formes. Les applications parallèles peuvent disposer de cette puissance de calcul comme si elle émanait d'une seule et même machine virtuelle. Accessibles au travers du réseau, les nœuds, constituent cependant, un ensemble très hétérogène. Cette hétérogénéité des ressources de calcul reste un point crucial pour exploiter de façon optimale les grappes et les grilles. Nous décrivons dans le premier chapitre, les solutions possibles, pour remédier à ce problème. Chaque nœud de calcul peut disposer de systèmes et de bibliothèques d'applications très différents. La technologie Java que nous avons mise en œuvre, permet d'aplanir cette difficulté. L'utilisation des machines virtuelles masque l'hétérogénéité du matériel et des différents systèmes présents sur les nœuds. L'autre avantage qu'offre Java est la programmation par objets et par composants. C'est dans ce contexte que nous avons conçu et développé la plate-forme logicielle ADAJ (Adaptative Application in Java). Pour faciliter le développement d'applications réparties à l'aide de cette plate-forme, nous lui avons adjoint un environnement de programmation complet.

Le second chapitre traite de l'efficacité de l'exécution des applications distribuées. Il montre que les performances sont fortement liées au choix de la granularité des tâches composant l'application, mais également aux conditions selon lesquelles ces tâches sont projetées sur le support d'exécution. Choisir une granularité appropriée reste une opération difficile. Le choix automatique de celle-ci, constitue une bonne solution pour résoudre ce problème. Il conduit à définir des applications auto-adaptatives qui exploitent de façon quasi-optimale, les architectures sur lesquelles elles s'exécutent. L'implantation des composants de l'application distribuée sur une plate-forme hétérogène, constitue un challenge supplémentaire. Les ordinateurs sont partagés avec d'autres utilisateurs qui captent une part imprévisible des capacités de traitement disponibles. D'autre part, la quantité de traitements requise pour les différentes tâches composant l'application que l'on veut faire exécuter, varie au cours du temps. Un équilibrage purement statique, n'est pas suffisant pour répondre aux exigences des performances attendues. Nous nous sommes par conséquent, tournés vers des techniques d'équilibrage dynamique de la charge. Ces dernières permettent de faire face aux variations de la charge et aux changements de configuration de la plate-forme. Notre solution est basée sur la mise en œuvre de mécanismes spécifiques, dans l'intergiciel ADAJ. Ces derniers permettent de réaliser un équilibrage de charge dynamique sur l'ensemble des machines virtuelles composant la plate-forme, en agissant sur la répartition des objets de l'application. Cette répartition s'appuie sur un nouveau mécanisme d'observation de l'activité des objets et des relations entre eux. Nous proposons également plusieurs algorithmes, pour effectuer un placement initial optimal des objets. Le but de ces algorithmes est de limiter les transferts ultérieurs des objets de l'application à travers la plate-forme, pour équilibrer les charges.

Le troisième chapitre aborde le problème du passage à l'échelle de la plate-forme ADAJ, vers les grilles de calcul. Les aspects hétérogènes et dynamiques deviennent encore plus prononcés dans ce cas que pour les grappes. L'analyse menée dans ce chapitre, montre que la différence entre ces deux entités repose essentiellement sur la manière dont sont gérées les ressources. La possession des ressources par l'utilisateur est un principe de base de la grappe, alors que les grilles sont caractérisées par le partage de celles-ci. Dans le premier cas, les

ressources sont gérées de manière centralisée et tous les nœuds coopèrent pour former une machine virtuelle unifiée. Les applications destinées aux grappes de calcul sont développées par et pour le propriétaire de la plate-forme, pour maximiser l'utilisation des ressources. Dans le second cas, chaque nœud est un site d'exécution qui possède son propre gestionnaire de ressources. Les machines utilisées pour exécuter une application, sont choisies du point de vue de l'utilisateur, pour maximiser les performances de celle-ci, sans se préoccuper des effets induits sur l'ensemble de la plate-forme. Cette différence fondamentale dans la gestion des ressources, nous a donc amenés à revisiter les concepts architecturaux développés dans ADAJ. Comme les premières versions de ce dernier reposaient sur le logiciel de communication RMI, nous avons d'abord vérifié que RMI passait bien l'échelle. En réponse à cette question, il a fallu lui adjoindre des fonctions de gestion du nouveau support d'exécution qu'était la grille. Ce travail a été réalisé suivant deux approches : l'approche de développement à l'aide des composants logiciels et l'approche de développement à l'aide de webservices. Nous avons également comparé ces deux approches avec celle basée sur le calcul en nuages (cloud computing) dans laquelle elles peuvent s'intégrer. L'utilisation de ces technologies ne suffit pas à passer l'échelle. Il a fallu également développer un système d'information basé sur les agents mobiles, pour pouvoir prétendre gérer convenablement une grille de type ouvert. Nous avons donc proposé la plate-forme SOAJA. (Service Oriented Adaptative Java Applications) qui reprend toutes les fonctionnalités d'ADAJ, en y ajoutant une couche de webservices et les agents mobiles. Pour finir, nous avons fait évoluer SOAJA, en y ajoutant des services de fouilles de données. Cela a donné naissance à l'architecture WODKA (Webservices Oriented Data Mining in Knowledge Architecture), qui offre une prestation plus large de services basés sur l'extraction de connaissances. Cette dernière est une plate-forme de type SOKU (Service Oriented Knowledge Utilities).

Pour conclure, nous traçons les perspectives de nos recherches dans le domaine des systèmes embarqués, puisque ceux-ci seront amenés à moyen terme, à inclure dans une seule puce, l'équivalent de la puissance d'une grille de calcul actuelle.

Chapitre 1

Quel environnement pour les applications distribuées sur grappe d'ordinateurs?

1.1 Le problème de l'hétérogénéité des grappes

1.1.1 Origines du problème

Les applications scientifiques actuelles, telles que la simulation, l'optimisation ou la fouille de données, doivent être résolues à l'aide de logiciels, souvent très gourmands en temps de calcul. De plus, les machines dédiées et les supercalculateurs restent trop chers et difficiles d'accès pour l'utilisateur courant. Déplacer l'exécution vers des systèmes moins chers, généralistes et composés par des ordinateurs à grande diffusion (stations de travail et logiciels libres de droit), se révèle donc une démarche parfaitement légitime.

Dans ce contexte, les grappes d'ordinateurs proposent une architecture évolutive qui offre très souvent un meilleur rapport performance/coût que celui proposé par les supercalculateurs traditionnels. Une grappe (cluster) peut se définir comme un ensemble de nœuds de calcul interconnectés par un réseau local, de manière à permettre l'exécution simultanée de plusieurs programmes séquentiels ou parallèles. Un nœud est un ordinateur qui peut comporter plusieurs unités de calcul (CPU), de la mémoire, éventuellement un espace de stockage, et une ou plusieurs interfaces réseau. Les grappes peuvent être interconnectées de sorte que la mémoire puisse être, soit partagée, soit distribuée. L'idée d'utiliser les grappes pour le calcul se montre assez séduisante. En effet, toutes les entreprises possèdent ce type de matériel, souvent pas exploité à pleines capacités, non seulement pendant les périodes creuses (nuits et week-ends), mais aussi pendant les heures de travail.

L'autonomie d'administration des différents sites d'une grappe de calcul implique souvent une forte hétérogénéité, tant du point de vue des ressources informatiques que du point de vue des méthodes d'accès à celles-ci. Cette hétérogénéité rend encore plus nécessaire, l'utilisation de standards et de logiciels ouverts. Elle peut se retrouver à trois niveaux :

- **Hétérogénéité des ressources de calcul :** en termes de puissance de calcul, il est possible de trouver aussi bien des stations de travail, des ordinateurs de bureau, des serveurs ou même des machines parallèles puissantes. La variété des processeurs, de leur architecture, de leur puissance de calcul ou de leur mémoire, participe à l'hétérogénéité de la grappe.

- **Hétérogénéité du réseau :** en termes d'interconnexion entre les ordinateurs, les liens de communication peuvent avoir des débits et des latences différents.

- **Hétérogénéité des logiciels :** les nœuds de la grappe peuvent posséder différents systèmes d'exploitation avec différentes versions. Les logiciels disponibles sur chaque nœud peuvent être aussi différents et installés à des emplacements de répertoire variés (compilateurs, bibliothèques de calcul).

1.1.2 Solutions possibles

Plusieurs solutions sont possibles pour masquer l'hétérogénéité du matériel et du logiciel présente dans les grappes de calcul. Toutes donnent l'image d'un système unifié (Single System Image) de ces plates-formes. Cette intégration en un système unique, peut être aussi bien réalisée par le matériel, par un système d'exploitation ou par un logiciel spécifique. Au niveau du matériel, elle est obtenue par exemple, à l'aide d'une mémoire distribuée partagée. Au niveau du système d'exploitation, un système de gestion de fichiers distribués unique du type NFS (Network File System) ou une gestion globale des ressources et de l'ordonnancement, permet d'aboutir à ce résultat. Enfin, la dernière catégorie concerne des environnements de programmation, essentiellement basés sur des bibliothèques de communication. Leur mise en œuvre donne également, la vision unifiée d'une plate-forme matérielle. Ces environnements sont nommés intergiciels, car ils se placent entre le matériel et les systèmes d'exploitation. Ils constituent ou ont constitué des standards de fait pour les environnements de calculs distribués. PVM [PVM11] et certaines implémentations de MPI [MPI11] en sont les exemples les plus connus. Nous plaçons nos recherches à ce niveau de l'intergiciel ou du middleware.

Cependant, si les intergiciels intègrent bien les grappes de stations de travail en une seule machine virtuelle, les programmer efficacement nécessite une maîtrise et une expertise certaines. Notre ambition est que l'ingénieur puisse s'impliquer uniquement dans son activité habituelle de développement, sans se soucier des tâches compliquées de parallélisation, qui ne sont pas a priori les siennes. De plus, les résultats en termes de performances de l'application distribuée qu'il a pu créer à l'aide des outils précédents, sont loin de pouvoir être garantis. L'obstacle majeur vient directement, de l'hétérogénéité de la grappe, des applications et des données. La mise en œuvre de ces premiers intergiciels n'est visiblement pas adaptée au développement facile des nouvelles générations d'applications auxquelles nous faisions référence. Les questions que nous nous sommes posées, sont alors :

1) Comment fournir de la puissance de calcul, disponible à la demande?

2) Comment supporter une gestion automatique du parallélisme et la rendre transparente au programmeur?
3) Comment fournir une certaine qualité de service (temps d'exécution convenable et tolérance aux pannes)?

4) Comment fournir un environnement de programmation facile à utiliser?

Dans le paragraphe 1.4, nous verrons quelles réponses nous donnons à ces problèmes, à travers le Projet ADAJ (Adaptative Distributed Application in Java).

1.2 Les systèmes à image unique

1.2.1 Propriétés attendues

Dans le but de fournir une interface globale au-dessus d'une grappe et de masquer son hétérogénéité et son aspect distribué, les systèmes à image unique (SSI ou Single System Image) ont donc été développés. L'objectif de ces systèmes est de donner l'illusion d'une machine unique aux administrateurs et aux utilisateurs, en proposant un accès global aux ressources processeurs, mémoires et disques. Dans ce contexte, le système met en relation les différentes ressources de la plate-forme matérielle et les applications des utilisateurs. Il peut être très complexe et composé de différents éléments avec une implémentation spécifique pour une grappe donnée. Il doit permettre un accès facile aux ressources, aussi hétérogènes qu'elles puissent être et quelque soit leur localisation. De façon générale, il gère et alloue les ressources au niveau de la grappe. Il doit gérer aussi l'authentification et la confidentialité et peut fournir des services de visualisation et de monitoring. De manière plus précise, notre cahier des charges pour les systèmes à image unique, englobe les problématiques suivantes :

1 Une utilisation transparente des ressources systèmes : en particulier aucune connaissance de l'architecture sous-jacente ne devrait être requise et une interface unique pour l'accès à la grappe devrait être disponible.

2 Une migration transparente de processus et l'équilibrage de la charge de calcul sur les différents nœuds de la grille. Le chapitre suivant aborde cette problématique.

3 Une fiabilité accrue et une grande disponibilité du système : le système doit être capable de poursuivre les calculs, même si un ou plusieurs nœuds tombent en panne ou se retirent de la grappe. Il doit donc présenter de bonnes propriétés de tolérance aux pannes.

4 Un temps de réponse global du système amélioré.

5 Un management du système simplifié : administration et contrôle.

Deux autres propriétés très importantes sont requises de la part des systèmes :

- L'indépendance par rapport à la plate-forme : cette vision unifiée, globale, ne doit pas être liée au nombre et au type de nœuds auxquels elle est associée.

- La scalabilité des performances : elle doit être proportionnelle aux nombre de nœuds. En particulier, si une ou plusieurs machines sont ajoutées à la grappe, l'efficacité globale (à charge constante) des calculs doit s'en ressentir en augmentant.

L'idée des systèmes SSI est donc d'encapsuler les ressources systèmes et de fournir une couche d'abstraction, de telle sorte que les composants les utilisant, voient ces ressources encapsulées comme un tout. De cette manière, les systèmes à image unique établissent une frontière logique qui dissimule la nature distribuée des ressources sous-jacentes. Cette frontière se retrouve à plusieurs niveaux : au niveau matériel, au niveau du système d'exploitation ou au niveau de l'application.

1.2.2 Les systèmes unifiés du point de vue matériel

Les ressources matérielles de bas niveau, comme les mémoires ou les systèmes d'entrées/sorties, peuvent être reliées par des circuits électroniques spéciaux pour fournir un espace unifié de mémoire ou d'entrée/sortie. Ainsi, des dispositifs particuliers, permettent de voir une grappe d'ordinateurs comme un seul et unique système de mémoire partagée. C'est le cas par exemple, des DSM (Hardware Distributed Share Memory) ou, par le passé, des DEC (Digital/Compaq Memory Channel). Ces dispositifs ont été conçus pour fournir une interconnexion fiable, efficace et puissante entre les nœuds de la grappe. Cette mémoire virtuelle globale est obtenue en faisant du mappage de tout ou partie de la mémoire physique sur de la mémoire virtuelle locale (appelée mémoire réflective). Hormis les exemples que nous venons de citer, il existe aujourd'hui peu de dispositifs basés sur des composants matériels qui sont utilisés dans le cadre des grappes de calcul. Les dispositifs existants ont tous en commun de mettre en œuvre des circuits spécialisés coûteux et adaptés à des applications spécifiques. C'est pour cela qu'ils sont plus particulièrement exploités par les constructeurs d'ordinateurs pour bâtir des supercalculateurs.

Cependant, la solution de la mémoire réflective commence à être utilisée, par des plates-formes virtuelles construites entre le matériel et le système d'exploitation. Elle met en œuvre des techniques qui permettent de restituer plusieurs environnements d'exécution à partir d'un même support matériel. Ainsi, des concepteurs comme Intel proposent des processeurs qui supportent la virtualisation de la mémoire mais aussi des systèmes d'entrées/sorties. Les composants VT-x et VT-i sont les premiers représentants de cette technologie de virtualisation et donnent une solution à ce problème à des processeurs comme le IA-32 ou le Titanium. La virtualisation est introduite par le biais d'un moniteur de machine virtuelle dont le rôle est de rendre accessible toute la plate-forme hardware par un ou plusieurs systèmes d'exploitation. Cependant, Intel avoue lui-même que malgré les perspectives prometteuses de cette technique, beaucoup de défis restent à relever, avant de pouvoir bâtir des systèmes efficaces basés sur ces processeurs. Les solutions proposées sont en effet, souvent trop complexes et par conséquent moins robustes. Elles nécessitent parfois même, la modification du code des systèmes d'exploitation.

D'autres recherches sont aussi menées, dans le but de concevoir de nouveaux composants matériels (hardwares). Ainsi, Liang Yong [YON08] a présenté en 2008, un dispositif développé à partir de techniques de mémoire virtuelle partagée (Distributed Virtual Machine Memory), qui fonctionne au-dessus d'une grappe constituée par des nœuds de type SMP (Symmetric Multiprocessor). Ce qui lui permet de distribuer la mémoire entre les différents nœuds et d'associer les avantages de chacune de ces architectures : bonne programmabilité pour les architectures de type SMP et meilleure aptitude au passage à l'échelle pour les systèmes du type mémoire distribuée. D'autres projets, comme InterWeave [TAN03], MOL [DEQ03], JuxMem [ANT05],ou Teamster-G [LIA05], offrent également cette possibilité de

disposer de mémoires virtuellement partagée et qui sont construites à partir de mémoires distribuées. Cependant pour intégrer toutes les propriétés requises et obtenir un vrai système à image unique, il reste encore beaucoup de travail à accomplir sur ce type de composants.

1.2.3 Les systèmes unifiés du point de vue des systèmes d'exploitation

Placer le système à image unique au niveau du système d'exploitation, signifie que c'est le noyau système lui-même qui est chargé de cacher la nature distribuée du hardware. Il faut qu'il fournisse aux sous-systèmes et aux applications, l'illusion d'une plate-forme unifiée. Il doit répondre avec la même efficacité aux appels systèmes quelles que soient leur provenance, que ce soit des sous-systèmes ou des applications. Les noyaux SSI sont nés à partir des systèmes issus des grappes de calcul haute performance du type Beowulf [BEO11]. Ces derniers étaient typiquement utilisés pour supporter des programmes parallèles de calcul dans des domaines tels que la prévision météorologique. Les calculs étaient initiés par un maître et les nœuds communiquaient entre eux pour les accomplir entièrement. Les systèmes Beowulf ont été ensuite améliorés. L'amélioration provient du fait que la grappe considérée dans le cas de ces systèmes, ne présentait pas auparavant, une image totalement unifiée de la plate-forme d'exécution. Ainsi l'accès aux données peut être lui aussi unifié, soit par des systèmes de fichiers soit par des bases de données. Les projets GFS [GFS11], OpenGFS [OPE11] ou Lustre [LUS11] proposent un accès parallèle à des systèmes de fichiers de données. En ce qui concerne les bases de données, Oracle propose aussi une solution qu'il appelle « Oracle Parallèle Server » (OPS, rebaptisé depuis Oracle 91 RAC).

D'un point de vue fonctionnel, un système d'exploitation de type SSI, doit exécuter efficacement des applications parallèles dans un environnement partagé avec d'autres applications séquentielles. Il faut donc qu'il puisse ordonnancer les tâches en provenance de ces deux types de programme. Il utilise pour cela des algorithmes d'ordonnancement basés sur le paradigme du "Gang scheduling". Dans ce fonctionnement toutes les tâches d'une même application parallèle sont regroupées et ordonnancées de manière concurrente sur les différents processeurs de la grappe. Ces tâches entrent alors en concurrence avec les applications séquentielles qui peuvent être exécutées en même temps sur chaque nœud. Les raisons avancées pour adopter dans ce cas, l'ordonnancement du type Gang, sont le partage efficace des ressources et la facilité de programmation.

1.3 La solution de l'intergiciel

1.3.1 Définition d'un intergiciel

Pour donner d'une grappe de calcul, l'illusion d'une plate-forme unifiée, une alternative est obtenue par l'emploi d'une couche logicielle appelée intergiciel (en anglais middleware). La notion de middleware n'est pas totalement nouvelle, puisqu'en 1968, P. Naur et B. Randell [NAU68] en jetèrent les prémisses. Le terme d'intergiciel quant à lui n'est apparu qu'un peu plus tard.

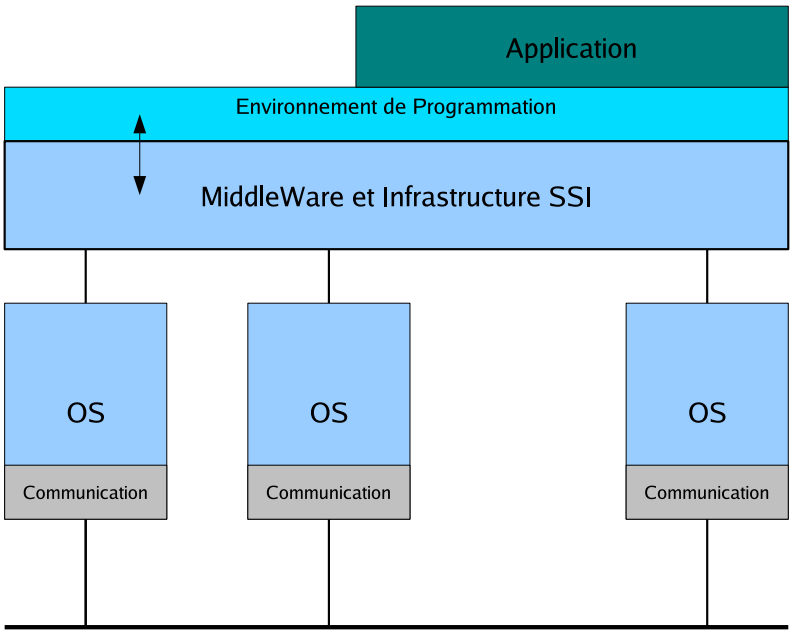

Figure 1 : Intergiciel ou middleware

Comme le montre la figure 1, l'intergiciel (infrastructure SSI) se situe entre l'application et une couche de communications et de services permettant l'accès aux ressources. Il permet à l'application de voir la grappe comme une seule ressource alors qu'elle s'exécute sur les différentes machines qui forment la grappe. Cette unification des ressources en une seule machine virtuelle, permet de s'abstraire de la couche de communication et des différents systèmes d'exploitation des nœuds de la plate-forme. Des concepts de plus haut niveau comme les services, les objets, les composants répartis ou les espaces partagés peuvent également être introduits. Grâce à une interface spécifique de type API (Application Programming Interface), l'intergiciel permet d'accéder de façon transparente à ces derniers. Il doit offrir en particulier les services de base, suivants :

- Traduction : ce service rend possible la communication entre machines mettant en œuvre des formats de données différents.

- Adressage : ce service identifie la machine serveur sur laquelle est localisé le service demandé et calcule le chemin pour y accéder. Il s'appuie généralement sur la mise en œuvre d'un annuaire centralisé ou décentralisé.

- Sécurité : ce service garantit la confidentialité et la sécurité des données grâce aux mécanismes d'authentification et de cryptage des informations.

- Communication : ce service permet la transmission sans altération, des messages entre le prestataire d'un service donné (serveur) et le demandeur (client). Il gère la connexion et la déconnexion au serveur, la préparation de l'exécution des requêtes et la récupération des résultats.

Les approches par intergiciel ne demandent pas la modification des systèmes d'exploitation pour faciliter l'intégration des nœuds dans la grappe. Les propriétaires des nœuds ne sont pas par conséquent, obligés de changer le système d'exploitation déjà installé et ils peuvent continuer à exploiter leurs propres applications. L'intergiciel repose sur l'association de deux modèles : un modèle de programmation et un modèle de communication. Le modèle de communication permet une gestion implicite ou explicite des communications, au niveau de l'application, par le modèle de programmation.

Le modèle de programmation définit la manière dont l'intergiciel doit être utilisé pour assurer les communications entre les composants logiciels. Il décrit par ailleurs, la manière dont il faut utiliser les composants, à l'aide d'interfaces de programmation d'application (API) et/ou de descriptions externes des interfaces (langages spécifiques et procédés d'annotation). Depuis la version 1.5 de Java, les annotations permettent d'assigner des attributs à de nombreux éléments du langage, y compris les classes et les méthodes Ce procédé a l'avantage de rendre les applications indépendantes de tout framework spécifique.

De très nombreux intergiciels présentent des modèles de communication et de programmation fortement couplés. Il commence à apparaître d'autres intergiciels réflexifs et multi-modèles, capables d'une part de modifier leurs propres comportements pour s'adapter aux changements extérieurs, et d'autre part de communiquer avec des composants développés à l'aide de plusieurs intergiciels différents. Nous ne nous sommes pas intéressés à ce type d'intergiciel puisqu'a priori le contexte applicatif est différent. En effet, il semble plus adapté à un contexte de mobilité.

1.3.2 Modèles de communications

Le modèle de communication permet de gérer les communications implicitement ou explicitement, au niveau de l'application. Il a un rôle crucial, puisque de lui dépendent en grande partie les performances, en terme de temps d'exécution. Beaucoup de systèmes ou logiciels de communication ont été proposés pour effectuer du calcul parallèle. Le pionnier de ces logiciels a été sans nul doute, le système d'appel de procédure à distance RPC (Remote Procedure Call) qui a été introduit avant les années 1980. Un appel de procédure à distance consiste à réaliser des appels de procédure entre des programmes distribués. Les paramètres des procédures permettent d'échanger des données entre l'appelant et l'appelé. Ce modèle permet d'appeler des fonctions qui s'exécutent à distance, comme si elles se trouvaient localement dans le même processus.

Le paradigme du passage de message (message passing) s'est ensuite développé avec l'introduction des machines parallèles à mémoires distribuées. Ce mode de communication est particulièrement bien adapté lorsque la structure et le timing des communications sont prévisibles, aussi bien du point de vue de l'émetteur que de celui du récepteur, et que ces derniers sont impliqués dans une opération de type collectif. Ce qui signifie que les communications sont explicites et n'imposent pas de structuration de l'application. Le passage de messages consiste à établir un canal de communication entre deux flots d'exécution pour y envoyer et y recevoir des données. La librairie MPI (Message Passing Interface) est, dans ce contexte, la plus utilisée. L'exécution d'une application implémentée à

l'aide de cette librairie, peut se montrer très problématique dans le cas de l'utilisation d'un grand nombre de nœuds de calcul composés de calculateurs multi-cœurs. Le problème est dû au nombre très important de tâches reposant sur des communications globales qu'engendre une application MPI faiblement couplée. Le passage de documents est un cas particulier de plus haut niveau du passage de messages où les données sont typées. Ce modèle est indépendant des protocoles de communications, et il est notamment utilisé par les Web Services (cf chapitre 3).

Basés sur les concepts des systèmes RPC et créés dans l'univers du langage Java, les systèmes utilisant les procédures RMI (Remote Method Invocation) [RMI11] ont ensuite fait leur apparition. Les applications distribuées, basées sur ce paradigme, ont des contraintes moins fortes en ce qui concerne le réseau que les applications MPI. En effet, ce mode de fonctionnement est bien adapté lorsque le système est faiblement couplé, ou lorsque les opérations de type collectif sont rares. Dans ce modèle de communication, l'accent est mis sur les calculs et les communications demeurent implicites. Des mécanismes d'asynchronisme, de passage d'arguments et de gestion des erreurs peuvent être alors mis en œuvre. L'appel des méthodes à distance est la transposition à la programmation orientée objet, du mécanisme des appels de procédure à distance du modèle RPC. Ces appels sont basés sur le protocole RMI dont les implémentations les plus fameuses sont Java RMI et CORBA (Common Object Request Broker Architecture) [COR11]. Cependant, si RMI et Corba cachent bien les échanges de messages derrière les appels de méthodes vers les objets distants, ils manquent tous deux de transparence pour être directement utilisés dans des systèmes distribués. Un logiciel comme JavaParty [JAV11] permet de combler cette lacune. Celui-ci implémente un environnement distribué qui est basé sur Java et qui est bâti sur une grappe de machines virtuelles (JVM) connectées par RMI. JavaParty gère automatiquement toutes les communications entre les objets globaux (définis comme accessibles à distance) de l'application. Il utilise un mot clé spécifique pour marquer ces objets que l'on veut partager au sein de l'application. Le traitement par un préprocesseur permet ensuite de traduire ce code en code Java/RMI. JavaParty fournit une simple interface pour que le programmeur puisse développer une application distribuée. Mais, il n'offre pas une transparence complète, car celui-ci doit gérer lui-même les objets locaux et les séparer des objets globaux.

Si les modèles de programmation synchrones de type RPC et RMI sont mieux intégrés aux langages objets, et donc plus faciles à mettre en œuvre, les modèles asynchrones permettent de gérer les communications à partir de, et vers, plusieurs objets ou composants. Le principal avantage de ces derniers, est de permettre l'établissement de communications sans connaître les autres composants de l'application. Néanmoins, le modèle RMI reste le moyen le plus simple pour les communications bidirectionnelles entre composants, car l'établissement du lien de retour dans le contexte asynchrone ne constitue pas une opération aisée.

1.4 Modèles de programmation

1.4.1 Programmations parallèle et distribuée

Les modèles de programmation s'appuient sur des paradigmes spécifiques. Pris au sens large, un paradigme est un modèle auquel s'apparente une famille d'exécutifs qui suivent certains critères bien définis. Ces paradigmes se classent soit sous l'égide de la programmation parallèle, soit sous celle de la programmation distribuée. Les définitions que nous en donnons

ne sont pas à prendre comme une classification stricte. Elles servent plutôt, à appréhender plus facilement les caractéristiques des modèles de communications présentés dans le paragraphe précédent.

La programmation parallèle a pour priorité la recherche de la haute performance. Les communications ont lieu à l'intérieur d'un ensemble prédéfini et connu de tous, de nœuds appartenant à une plate-forme. Elles se font très souvent à l'aide d'échange de messages. Il est courant de faire la distinction entre le parallélisme à mémoire distribuée et celui à mémoire partagée. L'utilisation de MPI correspond tout à fait à ce type de programmation asynchrone. Avec des communications asynchrones, l'initiateur peut continuer ses activités, immédiatement après que la communication a été lancée. Il peut ensuite vérifier le bon déroulement des opérations qu'il a lancées et attendre les résultats finaux. Les API dans ce contexte de la programmation parallèle sont conçues pour favoriser des implémentations « zéro-copie ». Des opérations collectives sont fournies pour une exécution optimisée des applications. Les applications qui utilisent le paradigme du passage de messages sont plutôt du type parallèle. Elles sont composées d'un ensemble de tâches qui s'exécutent en parallèle et qui peuvent être plus ou moins communicantes en fonction du degré de parallélisme choisi. Ces applications sont particulièrement sensibles à la qualité des réseaux sous-jacents. L'opération de parallélisation consiste à adapter la structure d'une application pour qu'elle tire profit de l'environnement parallèle dans lequel elle se trouve. Dans le cadre du paradigme de communication par messages, cette opération peut se révéler très complexe.

Dans la programmation distribuée ou répartie, la priorité est donnée à l'interopérabilité. Les connexions sont dynamiques. Elles sont souvent gérées lien par lien, en mode client/serveur, mais d'autres paradigmes sont bien évidemment possibles. Nous en décrirons quelques-uns dans ce paragraphe. L'interopérabilité est assurée entre des architectures pouvant posséder des systèmes et des logiciels différents. Ce mode de programmation correspond donc aussi à l'utilisation des procédures à distance de type RMI, qui est un cas typique de mise en œuvre de communications synchrones. Dans ce mode, l'initiateur doit attendre le retour des résultats pour pouvoir continuer ses activités. Ceci l'oblige par conséquent à se synchroniser avec l'exécution de la tâche qu'il a lancée. Cet obstacle peut être cependant levé par l'utilisation d'une programmation multithreadée qu'autorise le langage Java. L'autre inconvénient que présentent les applications basées sur l'utilisation de RMI, est l'absence d'opérations collectives. Nous avons apporté une solution à ce problème par la mise en œuvre des collections distribuées (voir le paragraphe 1.6).

1.4.2 Parallélismes implicite et explicite

Pour rendre le plus transparent possible le parallélisme d'une application, les détails internes de l'architecture matérielle qui supporte son exécution, doivent être cachés au programmeur. L'environnement doit fournir un support de programmation de haut niveau qui aide celui-ci, à produire et à exploiter le parallélisme potentiel de l'application. Deux manières sont possibles pour atteindre cet objectif :

La première de ces approches exploite le parallélisme implicite, à l'aide de langages et de compilateurs spécifiques. Le programmeur ne peut traiter et donc pas contrôler l'ordonnancement des calculs et le placement des données. De plus, les programmeurs rechignent à utiliser de nouveaux langages pour exploiter le parallélisme. Les compilateurs,

de leur côté, sont très souvent limités à des applications qui possèdent un parallélisme régulier comme le calcul dans des boucles. Si ces derniers se montrent très efficaces pour des architectures cibles telles que les multiprocesseurs à mémoire partagée, ils le sont moins pour des plates-formes à mémoire distribuée. L'environnement OpenMP [OMP11] est l'exemple type qui convient parfaitement aux architectures à mémoires distribuées. OpenMP permet en effet, d'exploiter le parallélisme à l'intérieur des boucles des programmes, en distribuant les différentes itérations à différents processeurs.

La seconde approche repose quant à elle, sur l'exploitation du parallélisme explicite par le programmeur ou par l'intergiciel. Ils en sont directement responsables et doivent alors prendre en compte des opérations comme la décomposition de l'application en sous-tâches et le placement de celles-ci sur les nœuds de la plate-forme. Gérer explicitement le parallélisme reste une tâche difficile pour un programmeur. Il faut éviter deux écueils qui peuvent se présenter à lui : les erreurs d'écriture du programme et/ou de l'application (erreurs qui impactent la validité de l'application) et les erreurs de gestion des performances (erreurs qui impactent la vitesse d'exécution). Le fonctionnement de RMI correspond à ce type de parallélisme. Les appels de méthode provoquent des appels dans le code de l'appelé (upcall) par l'intergiciel. Ce qui est donc une forme de réception de messages implicite. Les applications développées en MPI entrent également dans ce cadre de la programmation explicite. Les messages entrant pour la remise à jour des données et des informations, lors de l'exécution, doivent être recherchés explicitement par l'application. Le problème est de trouver la bonne fréquence pour cette mise à jour.

Ces deux types de parallélisme implicite et explicite peuvent parfois être combinés et donnent lieu à des méthodes hybrides. Ce dernier paradigme est notamment favorisé par l'apparition d'ordinateurs dédiés à la haute performance des calculs. Ce nouveau type d'ordinateur possède une architecture de grappe avec des nœuds multi-cœurs, telle que nous pouvions la décrire dans les paragraphes précédents. La programmation parallèle sur ces machines doit combiner une parallélisation de type mémoire distribuée entre les nœuds et une parallélisation de type mémoire partagée à l'intérieur de chaque nœud. Une programmation hybride permet d'exploiter pleinement, le potentiel de ce type de plate-forme SMP (Share Memory Processors). Chacun des modèles de communications (MPI et RMI) possède une solution pour aborder ce type d'architecture.

Ainsi, MPI est souvent associé à l'environnement OpenMP. Cependant, le couple MPI/OpenMP n'arrive pas à traiter efficacement le problème de l'exécution des applications sur des plates-formes comportant des processeurs multicœurs. Il ne procure pas des abstractions d'assez haut niveau et des recherches sur des nouveaux langages sont en cours. Ces langages sont des langages de type « dirigé par les données » (dataflow).

Le mappage de processus légers (threads) sur les différents processeurs physiques ou cœurs constitue la réponse du couple Java/RMI aux architectures à mémoire partagée de type SMP. Dans cette configuration, RMI continue à fournir les outils pour la partie programmation de la mémoire distribuée.

1.4.3 Paradigmes de programmation

Beaucoup d'applications de calculs distribués sont basées sur le paradigme client/serveur ou maître/esclave. Dans ce type de paradigme, les tâches communiquent entre elles à l'aide d'appels de procédure à distance. Pour chacune d'elles, la quantité de travail à fournir est connue à la compilation, car elle dépend de paramètres comme la quantité de données à traiter. Un exemple est le nombre de pixels à calculer dans le cas d'un lancer de rayon ou d'un calcul de radiosité effectué dans le cadre de la synthèse d'image. Les tâches clientes ne communiquent pas entre elles et rapportent les résultats calculés à la tâche maîtresse. Même si le parallélisme n'est pas inhérent à cette approche client/serveur, ce paradigme trouve de nombreux débouchés dans des applications importantes comme le repliement de protéines ou le séquençage du génome. Des plates-formes exploitent également cette approche : SETI@home proposé par l'Université de Berkeley (USA) pour la recherche de la présence d'extra-terrestres ou folding@home de l'Université de Standford (USA) pour la recherche contre le cancer.

Pour permettre une meilleure exploitation des possibilités de parallélisme des ressources distribuées d'une plate-forme telles que les grappes de calcul, des paradigmes de programmation plus complexes ont bien évidemment été introduits. Un premier groupe de paradigmes décrit la manière dont les calculs interagissent avec les données. Parmi ceux-ci on trouve les modèles SPMD (Single Processus Multiple Data ou Single Program, Multiple Data) et pipeline qui traduisent deux approches très différentes. Dans le premier modèle, chaque processus exécute le même code sur des données différentes. Les données de l'application sont réparties sur tous les processeurs et sont traitées de manière similaire. Le schéma des communications entre les processeurs est extrêmement structuré. Dans le modèle pipeline, les données produites à un étage inférieur sont consommées par l'étage suivant. L'application est découpée dans ce cas fonctionnellement. Au côté de ces deux modèles classiques, d'autres approches de répartition des calculs sont possibles. Voici l'exemple de trois approches qui nous paraissent très significatives, car elles sont très souvent mises en œuvre. La première d'entre elles s'appuie sur une opération de type collectif, la seconde sur un algorithme de type « diviser pour régner » et la troisième sur un algorithme de type Branch&Bound.

La première approche qui est parfois appelée paradigme de la ferme de tâches, étend directement l'approche maître/esclave. Elle généralise cette dernière en associant plusieurs esclaves à un seul et même maître. Le maître est responsable de la décomposition de l'application en sous-tâches qu'il doit distribuer entre les différents esclaves qui composent la ferme. Le maître doit également récolter les résultats de chacun des esclaves, au fur et à mesure qu'ils sont produits ou à la fin des traitements. Un esclave a donc un rôle très simple à jouer : attendre qu'une tâche lui soit affectée, la traiter et renvoyer le ou les résultats correspondants. Différentes politiques d'allocations de tâches entre les esclaves peuvent être mises en place : répartition acyclique, Round Robbin, etc. Ce paradigme de la ferme de tâches permet d'obtenir des gains appréciables sur l'exécution d'une application et de l'adapter très facilement en fonction du nombre de nœuds disponibles. La possibilité de répartir également le travail du maître, enlève l'obstacle du goulot d'étranglement que pourrait présenter une solution totalement centralisée. Ce paradigme est donc apte à passer l'échelle.

Contrairement à l'approche client/serveur, l'approche « diviser pour régner » partage récursivement les tâches en deux ou plusieurs sous-tâches. Le processus s'arrête lorsque les sous-tâches deviennent triviales à exécuter. A chaque étape, chacune des sous-tâches est

exécutée séparément et les résultats sont combinés eux aussi, récursivement, pour obtenir le résultat final. Souvent les tâches plus petites ne sont que des instances de la tâche originale. Les sous-tâches sont exécutées en parallèle. Cette opération peut cependant engendrer des communications importantes, si les sous-tâches ne sont pas indépendantes. Ce modèle « diviser pour régner » conduit à des systèmes structurés hiérarchiquement car les tâches sont créées par des subdivisions récursives. Le graphe des tâches qui est engendré est structuré de la même manière. Il est donc hiérarchique et peut être exécuté de façon satisfaisante avec des bonnes propriétés de localités pour les communications sur des plates-formes adéquates. Ce paradigme est donc particulièrement adapté pour des plates-formes d'ordinateurs structurées de manière arborescente et pour des applications à fort parallélisme de données comme la classification de données.

Le paradigme du Branch&Bound comporte un large spectre d'algorithmes, qui ont tous l'objectif de rechercher une solution optimale comprise dans un ensemble de solutions possibles. La taille de cet ensemble est typiquement exponentielle par rapport aux données initiales d'un problème. Considérons par exemple, le problème du voyageur de commerce qui est très connu sous le nom de TSP (Traveling Salesman Problem). L'ensemble des itinéraires possibles pour un voyageur devant visiter 23 villes différentes est de 22 !, soit autour de 1,6 x 10^{15}. Cet espace de solutions possibles est progressivement partitionné (branching) pour former un arbre. Chaque nœud est associé à une solution partielle réalisable. Poursuivre le processus en descendant dans cet arbre amène peu à peu des solutions de plus en plus complètes et de moins en moins nombreuses. Dans le cas du Branch&Bound, les nœuds de l'arbre sont simplement des tâches qui sont divisées en sous-tâches plus petites. Le processus de subdivision s'arrête lorsque la sous-tâche est assez petite pour être exécutée par un seul nœud de la plate-forme. L'approche maître/esclave est un cas limite de ce paradigme. Pour généraliser l'arbre dynamique créé dans le cas des algorithmes de type Branch&Bound, il suffit d'associer aux algorithmes du type maître/esclave, un arbre dont la profondeur de décomposition est fixe. Cependant l'implémentation d'un algorithme de type Branch&Bound, reste plus complexe que celle de type maître/esclave, car l'arbre de décomposition n'est pas équilibré dans son cas. Cette irrégularité est due à un processus d'élagage de l'arbre des solutions (bounding) qui dépend de l'instance du problème et qui est donc connu seulement à l'exécution. L'implémentation d'algorithmes de type Branch&Bound possédant un caractère auto-adaptatif est assez compliquée à réaliser. Ce qui rend difficile leurs passages à l'échelle sur des plates-formes possédant un très grand nombre de nœuds.

1.5 Etat de l'art

1.5.1 Exemples de systèmes SSI

Le développement des systèmes d'exploitation de type SSI, n'est pas récent. Il en existe un certain nombre qui ne sont plus maintenus : Amoeba [TAN99], Sprite [SPR11], et Nomad [PIN99]. Voici l'exemple de quatre autres systèmes qui ont connu des destinées diverses. Ils sont soit arrêtés, soit encore utilisés, soit intégrés dans de nouvelles plates-formes :

OpenMosix [BAR03]: en 1981, une Equipe de Recherche de l'Université de Jérusalem développait un système appelé MOS (Multi-computer Operating System) qui tournait sur Vax/BSD. Ce système s'est ensuite appelé Mosix, lorsqu'il a été porté sur le système Linux en 1999. En 2002, à la suite de divergences sur le futur du produit, le système OpenMosix a

été créé à partir du code source de son prédécesseur. L'objectif initial de Mosix était de proposer un ordonnancement global des processus dans la grappe en faisant migrer certains d'entre eux pour équilibrer la charge. D'autres composants permettant de globaliser les ressources ont ensuite été rajoutés, comme par exemple un système de fichiers global. Le développement d'OpenMosix a été arrêté en 2009.

OpenSSI (Open Single System Image) [WAL04] : c'est un système qui a été développé à partir de l'année 2001 pour fonctionner dans un environnement Linux. Sa base provient essentiellement du logiciel NoNStop Cluster [NON11]. Ce logiciel développé par le Constructeur Hewlett Packard et repris ensuite par d'autres comme Compaq Proliant, proposait de construire une grappe au-dessus du Système UnixWare. Aujourd'hui, nous retrouvons encore ce système qui a été intégré avec des distributions OpenSource de Linux comme Fedora, Debian ou RH9. Comme pour OpenMosix, certaines fonctionnalités de gestion de ressources ont été ajoutées au noyau primitif. Ainsi par exemple, grâce à un système de gestion de fichier (Cluster File System) un processus qui a été migré, est assuré de retrouver le même montage de fichier que celui qu'il avait sur le nœud d'origine. OpenSSI travaillant au niveau des processus, aucune modification de l'application n'est donc nécessaire et en particulier, il n'a pas besoin de librairie de parallélisation. En revanche, pour rendre possible les migrations, il est nécessaire de modifier le noyau Linux. De plus, si l'équilibre de charge se fait de manière transparente, il ne peut se faire qu'au niveau du processus (granularité forte).

Kerrighed [MOR04] : Kerrighed est un autre exemple de système d'exploitation à image unique qui étend les fonctionnalités et les interfaces de Linux à l'échelle de la grappe d'ordinateurs. Le projet a été initialisé en Octobre 1998, à l'INRIA de Rennes. Appelé à l'origine Gobelins, ce système se présente sous la forme d'un logiciel libre sous licence GPL. Il supporte les deux modes de programmation parallèle que sont la programmation par mémoire partagée et le passage de message (cf paragraphe suivant). Cet objectif est atteint par l'implémentation d'un ensemble de services distribués qui fournissent une gestion globale et dynamique de toutes les ressources physiques (mémoires, disques et interfaces réseaux), et par l'utilisation des concepts génériques suivants :

- Processus fantômes : utilisés pour la migration des processus et des points de reprise.
- Flux dynamiques : le mécanisme des sockets de Unix est utilisé pour la migration à plusieurs niveaux de communication, comme TC/IP ou Myrinet.
- Conteneurs : utilisés pour la mémoire partagée distribuée et un cache global.

Aujourd'hui la version grappe du système XtreemOS **[MOR07**] développé dans le cadre d'un Projet Européen, est basée sur l'open source de Kerrighed.

Plurix [GOE04] : Plurix est un système natif d'exploitation basé sur la programmation Java, qui a été développé au Brésil depuis 1982, par plusieurs établissements dont l'Université de Rio de Janeiro. Les communications entre les nœuds de la grappe, s'effectuent au moyen d'objets partagés dans une mémoire distribuée. Cette mémoire est organisée en pile de stockage distribué (Distributed Heap Storage) qui contient le code et les données. La consistance de la mémoire est garantie par la mise en œuvre d'opérations de type transactionnel. Plurix introduit des vecteurs de transactions, dont il peut plus facilement assurer la migration que par l'utilisation de processus. Ce qui lui permet de pouvoir proposer

des solutions assez efficaces pour l'équilibrage de charge et l'ordonnancement des applications qui l'utilisent. Néanmoins, l'utilisation de Plurix nécessite l'emploi d'un compilateur Java qui lui est propre. D'autre part, comme dans tous les systèmes de cette catégorie, y compris le précédent système Kerrighed que nous venons de décrire, le noyau système est modifié pour pouvoir relancer des applications qui se seraient mal déroulées.

Aujourd'hui, deux systèmes sont les héritiers de Plurix.

- Mulplix est un système d'exploitation de type Unix. Il supporte un parallélisme de programmation à grain moyen à l'aide de la mémoire partagée Multiplus. Cette mémoire supporte jusque 1024 noeuds et 32 GBytes d'adresses mémoire globale.
- Tropix est un système temps réel à mode préemptif pour grappe de PC, de type Linux.

Les améliorations des grappes SSI se classent en général, suivant trois schémas :

- extensions extérieures au noyau système pour la gestion ou la haute disponibilité de la grappe,
- utilisation de « patchs » systèmes pour ajouter des couches ou de nouvelles fonctions,
- extensions des sous-systèmes à l'intérieur du noyau.

Dans tous ces cas, la solution présente l'inconvénient de modifier le noyau système. De plus, posséder juste une image unifiée de la grappe d'ordinateurs, ne suffit pas pour considérer un grand nombre de nœuds, en particulier, si des propriétés comme la scalabilité ou la disponibilité veulent être garanties. La solution de placer l'image unique au niveau des systèmes d'exploitation, ne pourra être par conséquent efficace, qu'avec l'ajout de librairies de programmation.

1.5.2 Exemples d'intergiciels

Les intergiciels doivent fournir au minimum des mécanismes permettant de soumettre des applications à la grappe, de trouver des ressources, de déployer une application et de récupérer les résultats produits. Comme tous les systèmes unifiés, ils masquent la distribution des ressources à l'utilisateur. Ainsi les fonctionnalités d'allocation de ressources et d'interface utilisateur sont résolues, du moins partiellement, au sein de ces systèmes.

Condor [BAS99], THA05] : Condor est l'un des premiers intergiciels qui sont apparus dans les années 1980. Développé dans l'Université du Wisconsin, il a été développé pour faire des calculs très intensifs (high throughput computing). Il fonctionne sous le principe de vol de cycle, c'est-à-dire qu'il essaie d'utiliser au maximum, les ressources CPU inutilisées au sein des nœuds d'une grappe de calculateurs de bureau. Il utilise un système de publication dans lequel, chaque machine participante va exposer ses capacités de calcul et ses ressources disponibles. La distribution de tâches est ensuite réalisée par un gestionnaire central. Il utilise un mécanisme de bac à sable (sandbox) pour isoler les applications et éviter ainsi les problèmes causés par des codes erronés ou malveillants. Un mécanisme de point de reprise permet de déplacer du code. Comme point négatif, Condor est conçu pour des environnements où les utilisateurs disposent de machines puissantes et il n'alloue des travaux que sur des machines individuelles, sans aucune possibilité d'agrégation de plusieurs ressources entre elles. Son autre principal défaut est une trop grande réactivité à l'utilisation d'un nœud par

une tierce personne.

Dans la même lignée que Condor, IBM a développé **Entropia [CHI03**], un système professionnel, destiné au monde de l'entreprise. Le système Entropia exploite des machines virtuelles spécifiquement dédiées aux systèmes microsoft (Windows NT, 2000 et XP). Il agrège toutes les ressources de calcul (PC), en une entité virtuelle logique. L'architecture est divisée en trois couches : une couche correspondant à la gestion physique des nœuds, une couche qui ordonnance les travaux et enfin une couche relative à la soumission de ces travaux. La couche d'ordonnancement réalise la projection des travaux sur les différents nœuds et leur ordonnancement au cours de l'exécution. L'utilisateur peut directement interagir sur cette couche, au travers d'interfaces dédiées. Les machines virtuelles sont alors responsables du lancement et de la sureté des exécutions des travaux demandés.

Unicore (Uniform Interface to COmputing Resources) [ERW01] : Unicore provient d'un projet qui avait pour objectif d'exploiter les différents centres de calcul allemands, comme un seul calculateur. Il permet d'ordonnancer les différents travaux (jobs), de manière indépendante de la plate-forme. Ces travaux sont représentés sous la forme d'objets abstraits (Abstract Job Objects). Ces derniers sont des collections de classes Java contenant les informations nécessaires pour exécuter les travaux en mode batch, sur les nœuds de calcul. Unicore propose pour accéder aux fonctions de ce calculateur virtuel, des interfaces graphiques intégrant des fonctions de haut niveau. Mais, il ne propose pas la possibilité de monitorer ces travaux et par conséquent ne supporte pas d'exécution dynamique.

Legion [LEG11] : Ce projet américain est également relativement ancien puisqu'il date des années 1990. Il propose également d'obtenir un superordinateur virtuel en agrégeant toutes les ressources d'une grappe. Son originalité provient de son approche orientée objet : toutes les ressources matérielles et logicielles, les fichiers et les applications, sont des objets qui appartiennent à des classes. Il spécifie un certain nombre d'objets de bas niveau qui peuvent être modifiés ou étendus par l'utilisateur. Il supporte des langages parallèles et des bibliothèques parallèles telles que PVM ou MPI.

Comme nos propos ne concernent ici que les grappes de calcul, nous évoquerons plus tard dans ce mémoire, le cas d'autres systèmes qui se situent dans cette lignée, mais qui visent plus particulièrement des plates-formes à plus grande échelle.

1.5.3 Exemples d'environnements de programmation

Le but des modèles de programmation de haut niveau pour le calcul distribué est de fournir des environnements de programmation les plus simples que possible à utiliser, mais permettant une exécution efficace des applications sur des plates-formes hétérogènes. Nous donnons ici deux exemples assez caractéristiques des environnements de programmation. Le premier Satin [NIE10], se focalise sur deux classes de problèmes importants que nous avons décrits dans le paragraphe concernant les paradigmes de programmation et qui s'appuient respectivement sur le paradigme client/serveur et sur le paradigme « diviser pour régner ». Ce modèle est surtout utilisable pour des architectures matérielles de type hiérarchique. Le second exemple, Proactive [PRO11], est plutôt un modèle générique utilisable par tous types d'applications distribuées.

Satin [NIE10] est un famework en Java qui permet aux programmeurs de paralléliser des applications basées sur le paradigme « diviser pour régner ». Son but est de les libérer d'une gestion manuelle pour rendre plus efficace l'exécution de leurs applications. Il met en œuvre des objets partagés, des exceptions asynchrones et des mécanismes d'interruption de processus. Il fonctionne sur de larges plates-formes d'ordinateurs sans mémoire commune et a pour cible des applications distribuées à grain fin. Satin ne s'intéresse pas particulièrement au paradigme de la ferme de tâches ni à celui, très simple, de client/serveur développé dans les applications parallèles du type SETI@home. Mais ces applications peuvent également être exécutées avec Satin.

Satin est implémenté au-dessus de l'environnement de programmation Ibis [NIE02]. Ce dernier est un environnement basé sur Java. Il est composé d'une librairie d'outils de communication RMI et d'une variété de modèles de programmation surtout basés sur des protocoles d'échange de messages de type MPI. Satin étend Java à l'aide de deux primitives pour paralléliser les programmes conventionnels qui utilisent les threads : la primitive « spawn » pour subdiviser les calculs en sous-tâches et la primitive « sync » pour bloquer l'exécution jusqu'au retour des résultats des sous-tâches. Les méthodes qui ont vocation à être parallélisées sont caractérisées par un marqueur spécial d'interface qui étend l'interface « Satin.Spawnable ». Elles doivent implémenter cette interface et le résultat de l'invocation de cette méthode doit être stocké dans une variable locale. Satin utilise alors des techniques de compilation et d'instrumentation de bytecode Java, pour transformer les méthodes « spawnable » et les appels de synchronisation, en tâches exécutables qui lui sont propres.

Proactive [PRO11] est une suite logicielle destinée au monde des applications parallèles, distribuées et multithreadées en Java. Proactive fournit un grand nombre de services comme la migration des objets d'un nœud de calcul à l'autre, la sécurité, la communication de groupe. Proactive met à disposition des outils pour réaliser des communications de groupe qui sont basées sur les invocations de méthode Java. Comme Proactive est un modèle de communication explicite, le programmeur doit gérer lui-même les optimisations pour une exécution efficace. Proactive supporte la migration des objets entre les JVM. S'il fournit les outils nécessaires à la migration et la tolérance aux pannes, c'est à l'utilisateur de gérer lui-même ces aspects. Proactive se présente sous la forme d'une bibliothèque de type API qui cache la complexité des communications et des mécanismes d'introspection qu'il met en œuvre. Cette API permet également de transformer des classes Java classiques, en application Proactive. Une application de ce type est composée d'un certain nombre d'entités mobiles qui sont appelées objets actifs. Un objet actif est un objet dans lequel l'appel et l'exécution d'une méthode sont asynchrones. Il reçoit des requêtes par un stub qui fait le lien entre lui et l'utilisateur. A chaque objet actif est associé un thread dans lequel il exécute au fur et à mesure les requêtes qui lui sont adressées : il est actif. Lorsque l'exécution de la requête est terminée, le résultat est retourné. Chaque objet actif dispose donc, de son propre thread de contrôle, et d'un point d'entrée racine, par lequel les fonctionnalités de l'objet sont accessibles par les objets ordinaires. Les appels de méthodes sur les objets actifs sont synchronisés par la mise en œuvre de mécanismes « d'attente par nécessité » qui bloquent l'appelant jusqu'au retour du résultat. Pour rendre l'application indépendante de la plate-forme d'exécution, Proactive introduit la notion de nœud virtuel. L'inconvénient, c'est que ceci introduit une dépendance de la description de l'application avec la description du déploiement, qui empêche un déploiement systématique de l'application.

1.6 Résumé de nos contributions

Le projet ADAJ (Applications Distribuées Adaptatives en Java) répond à la problématique posée par les applications distribuées devant être exécutées sur des grappes hétérogènes de calcul. ADAJ offre un environnement d'exécution et des outils pour développer des applications distribuées fonctionnant sur cet environnement (référence : [7]).

La mise en œuvre du langage Java et des machines virtuelles qui lui sont associées permet à ADAJ, de fournir une image unifiée de la plate-forme. L'utilisation de ce couple permet de s'abstraire du contexte varié fourni par les différentes configurations du hardware, les différents systèmes d'exploitation, versions et librairies qui sont utilisés sur chacun des nœuds d'une plate-forme de calcul hétérogène. Comme le permet le langage Java, ADAJ gère l'hétérogénéité d'un environnement distribué au niveau conceptuel de l'application. Il offre une interface de programmation (API) qui facilite le développement d'applications parallèles et distribuées. ADAJ fait également bénéficier l'utilisateur d'autres avantages majeurs qu'apportent les approches centrées autour de Java. Ceux-ci concernent, en particulier, la mobilité du code et la sécurité obtenue lors de l'exécution de celui-ci, dans les machines virtuelles.

Au niveau de la conception, ADAJ propose des solutions basées sur une bibliothèque de développement d'applications parallèles et distribuées. L'expression du parallélisme en ADAJ met en œuvre des collections distribuées qui regroupent des objets fragmentés et distribués. Les fragments sont distribués et des traitements parallèles peuvent être entrepris sur ces derniers. L'utilisateur a la possibilité d'activer les traitements selon un paradigme de calcul de type MIMD (Multiple Instruction Multiple Data). Ceci est rendu possible grâce à l'utilisation conjointe d'un parallélisme de données amené par les collections distribuées et d'un parallélisme de tâches basé sur des appels asynchrones de méthodes. La taille des fragments fixe le degré du parallélisme Ce choix peut être repoussé au moment de l'exécution. L'indépendance vis-à-vis des contraintes de localisation explicite des objets Java et de leurs possibilités de migration ultérieure pendant l'exécution, facilite l'utilisation de cette bibliothèque. Cet aspect concernant la conception d'application pour ADAJ a été en particulier, traité dans la thèse de Violetta Felea [FEL03].

Au niveau de l'exécution, ADAJ fournit des mécanismes d'équilibrage de charge qui peuvent dynamiquement adapter l'exécution aux modifications du support d'exécution et aux évolutions des calculs. (références [6] et [18] à [22]) L'efficacité d'exécution en ADAJ repose sur un mécanisme d'observation, qui permet d'acquérir une connaissance du comportement du traitement. Les objets distribués d'une application sont observés et redistribués dynamiquement en fonction de certains critères d'attraction qui ont été définis dans ADAJ. Cet aspect concernant l'exécution d'application en ADAJ a été traité dans les thèses d'Amer Bouchi [BOU03] en ce qui concerne le système d'observation et de Violetta Felea, en ce qui concerne le système de gestion de ces informations. Le chapitre suivant, nous permettra de développer plus en profondeur, les aspects liés à l'exécution des applications avec la plate-forme ADAJ.

Chapitre 2

Efficacité de l'exécution des applications distribuées

2.1 Granularité des applications distribuées

2.1.1 Le problème du choix de la granularité

Le calcul distribué a été introduit depuis bon nombre d'années pour pouvoir résoudre le plus rapidement possible des problèmes scientifiques (biologie, météorologie) très gourmands en temps de calcul, sans exiger l'acquisition d'ordinateurs très puissants et coûteux. Les deux principales caractéristiques d'une application distribuée sont l'absence d'un système commun et le non-partage d'un même espace d'adressage en mémoire physique, par les différents processus qui la composent. Une application distribuée utilise des ressources réparties, qui fonctionnent concurremment pour accroître la puissance de calcul disponible pour sa résolution. De cette façon, le calcul distribué peut prétendre exploiter pleinement la puissance des plates-formes d'exécution telles que les grappes ou les grilles de calcul.

Cependant la parallélisation efficace d'une application distribuée sur de telles architectures, reste un problème difficile. Une application est définie par un ensemble de programmes qui sont, eux-mêmes, composés par des tâches ou des processus à exécuter. De manière traditionnelle, paralléliser un programme consiste donc à définir chacune de ces tâches, qu'elles soient concurrentes ou non, ainsi que les données nécessaires à leurs exécutions. Dans ce contexte, le grain de calcul correspondant à chaque tâche, représente un bloc unitaire qui sera évalué localement et de manière séquentielle par un seul processeur. De façon analogue, on peut définir le grain de l'application comme la partie de l'application qui s'exécute sur un nœud de la plate-forme d'exécution (processeur, grappe ou multiprocesseur). La performance d'une application parallèle dépend beaucoup du choix de ce grain. Deux points de vue classiques de l'algorithmique parallèle sont possibles pour choisir la granularité. Une première approche fixe le grain en fonction du nombre de ressources qui pourront être disponibles lors de l'exécution de l'application. L'inconvénient de cette approche est qu'elle ne prend pas en compte les possibles variations du nombre et de la puissance des nœuds. La deuxième approche, plus théorique, fixe le grain pour minimiser le temps d'exécution sur un nombre infini de processeurs, tout en conservant un nombre total d'opérations proche du meilleur algorithme séquentiel. L'inconvénient de cette dernière approche est que, lors d'une exécution effective sur un nombre sans cesse restreint de ressources disponibles, le placement et l'entrelacement de toutes les tâches peuvent entrainer des surcoûts importants.

2.1.2 Choix automatique de la granularité

L'approche idéale pour concevoir un programme distribué serait donc de le rendre le plus indépendant possible de la plate-forme d'exécution. Cette indépendance doit être effective à tous niveaux : nombre de processeurs, réseaux et systèmes. Cependant, le programmeur doit tout de même avoir connaissance des possibilités offertes par l'environnement de programmation qu'il utilise. Mettre en œuvre une application avec un environnement de type Java/RMI est vraiment différent du développement de cette application avec une approche du type passage de messages, comme peut l'offrir par exemple, l'environnement MPI. De la même façon, le programmeur doit avoir connaissance d'autres éléments et caractéristiques de l'environnement auquel est destinée son application, comme les mécanismes de tolérance aux pannes ou d'équilibrage de charge. Si certaines fonctions ne sont pas fournies dans cet environnement, il doit veiller à y suppléer lors du développement de l'application. Toutes ces contraintes peuvent donc limiter le programmeur dans la conception, mais également dans l'efficacité, d'une application distribuée. Ainsi, le choix par le programmeur, de la granularité et du degré de parallélisme dans les calculs, peut avoir ultérieurement de lourdes conséquences sur l'efficacité de l'exécution de l'application. Celle-ci est fortement dépendante des communications, tributaires elles-mêmes du choix de la granularité. Si le nombre de tâches créées est inférieur au nombre de ressources disponibles, l'application ne bénéficiera pas de toute la puissance potentielle de calcul que peut offrir la plate-forme. Si, en revanche, comme nous l'avons déjà fait remarquer précédemment, le nombre de tâches créées est supérieur, le placement de ces nombreuses tâches entraine des surcoûts, tant du point de vue de la quantité de communications que de celui de la synchronisation entre tâches. Si le nombre de tâches créées est très élevé, ces surcoûts augmentent beaucoup le temps d'exécution globale de l'application. Cela prouve bien que le choix de la granularité du parallélisme repose fortement sur le ratio existant entre les performances de calcul et celles des communications.

Il est ainsi difficile pour le programmeur de développer une application avec une granularité optimale pour une plate-forme dédiée, si celle-ci est susceptible d'évoluer continuellement. Quand l'application est très dépendante de la taille des données, par exemple, il est possible avant le début de l'exécution, de définir des paramètres pour adapter la granularité en fonction de la taille des données. Cette adaptation peut cependant ne plus être optimale lors de l'exécution, si la taille des données à traiter augmente trop avec les calculs. Il est donc nécessaire de pouvoir modifier automatiquement le degré de parallélisme durant l'exécution. L'exploitation efficace d'une plate-forme de type grappe d'ordinateurs, passe par conséquent, par le développement de solutions capables d'adapter dynamiquement le grain d'exécution des calculs pour une application donnée, en fonction des variations de la plate-forme, dans sa constitution et ses performances.

2.2 Auto-adaptativité des applications distribuées

2.2.1 Définition de l'auto-adaptativité

Afin de tirer entièrement profit de toute la puissance disponible et de la flexibilité des plates-formes telles que les grappes ou les grilles de calcul, les applications doivent être en mesure

de présenter un degré satisfaisant de parallélisme de calcul et de données. Pour pouvoir atteindre cet objectif de performances optimales, il est intéressant d'utiliser des applications ou des algorithmes qui possèdent un caractère auto-adaptatif. Une application est dite auto-adaptative, si elle est capable de changer automatiquement son comportement en fonction des variations de son environnement d'exécution. La répartition de ses différents composants exécutés sur la plate-forme varie dans le temps, en fonction de la disponibilité des ressources et des variations de la puissance disponible des nœuds. L'auto-adaptativité peut se faire également en fonction des variations des états du calcul et des ressources des données manipulées.

L'auto-adaptativité s'appuie sur des stratégies de décision qui déterminent le comportement général de l'exécution de l'application sur la plate-forme. Ces stratégies peuvent ou non dépendre de la configuration matérielle de l'environnement (nombre de processeurs ou de cœurs, hiérarchie des caches). En particulier, une application et un algorithme sont indépendants des ressources lorsque les stratégies de décision n'utilisent aucun de ces paramètres de configuration et qu'elles peuvent changer en fonction des seules variations de l'environnement de calcul et de la charge induite par les autres utilisateurs.

2.2.2 Gestion de l'auto-adaptativité au niveau de l'application

Les différentes parties d'une application auto-adaptative doivent être en mesure de s'exécuter en parallèle avec un grain de calcul adéquat pour leur distribution sur la plate-forme. Pour cela, elles doivent évaluer automatiquement les caractéristiques sous-jacentes du support et se reconfigurer dynamiquement en fonction de l'état et de l'évolution de ce dernier, pour maintenir un niveau de performance requis à l'avance. Cette reconfiguration dynamique implique qu'une telle application puisse redéployer sur la plate-forme, pendant l'exécution, les différentes parties qui la composent. L'application doit donc posséder une aptitude à changer son degré de parallélisme ou son déploiement, au gré de l'évolution des calculs et des variations de configuration au sein de la plate-forme. Cette adaptativité traduit des actions et des corrections sur des déséquilibres de charge qui peuvent, bien sûr, être pris en compte au niveau des applications elles-mêmes. Généralement face à des quantités de données énormes à traiter, et face à une exigence de ressources intensives de calcul, ces applications ne disposent que d'un nombre limité de moyens, en termes de processus coopérants. De plus, ces applications ne possèdent que très peu d'informations sur l'environnement dans lequel elles sont mises en concurrence entre elles. Tous ces éléments, auxquels s'ajoute une certaine complexité pour le développement d'applications ad hoc, rendent la gestion d'une haute dynamicité de la plate-forme très difficile au niveau de l'application.

2.2.3 Gestion de l'auto-adaptativité par un middleware

Gérer les reconfigurations dynamiques sur de grandes plates-formes hétérogènes ne devrait pas être directement du ressort du développeur de l'application distribuée. Ce travail devrait être le plus transparent possible pour lui. Les différents niveaux de reconfiguration qui permettent de gérer l'auto-adaptativité de l'application en réaction aux modifications de la plate-forme d'exécution, incorporent des opérations telles que le lancement, l'arrêt, le redémarrage ou la migration des tâches qui composent les différentes parties de l'application. Ces opérations exigent de la part du programmeur de maîtriser des techniques complexes liées à des langages ou à des modèles de programmation sophistiqués. Ces modèles

fournissent des outils spécifiques pour permettre cet usage. Ils fournissent des abstractions de hauts niveaux qui permettent de structurer l'application en entités parallèles ou distribuées qui coopèrent pour accomplir des tâches spécifiques.

Il semble donc plus opportun de confier la tâche de la gestion de la plate-forme d'exécution elle-même, à un intergiciel qui sera, lui, mieux à-même de déployer efficacement l'application et de la redéployer le cas échéant de manière transparente. Alors que les modèles de programmation permettent de structurer convenablement l'application parallèle et/ou distribuée, les mécanismes du middleware, quant à eux, doivent encapsuler tous les détails intrinsèques à la gestion de la plate-forme d'exécution et à la politique de reconfiguration. Modèles de programmation et middleware travaillent ainsi en symbiose et permettent à l'application de répondre convenablement aux décisions de reconfigurations du middleware. Les reconfigurations de type fonctionnel, comme la migration des entités de l'application ou des changements de la granularité de ces entités, dépendent donc fortement du modèle de programmation et du langage qui ont été choisis pour le développement de l'application.

Pour aider le middleware à fonctionner correctement, il est nécessaire de disposer d'un ensemble d'entités autonomes qui puisse fournir assez d'informations sur les états internes de l'application et sur leurs interconnections. Ces informations concernent les ressources courantes auxquelles l'entité est liée, les performances attendues, le profil des performances sur une période de temps donnée et des prédictions de performances pour le futur. Si ces informations sont disponibles, le middleware pourra les utiliser pour automatiser le processus de l'auto-adaptation de l'application dans un environnement d'exécution dynamique. Le middleware évite ainsi au programmeur de devoir gérer des cas de reconfiguration très complexes. En particulier, ce dernier n'aura plus à décider d'avance quand et où il faudra faire migrer telle ou telle partie de l'application, et quelle entité aura besoin d'être reconfigurée.

2.3 Mappage des applications distribuées sur les grappes

2.3.1 Problèmes posés par les variations de la charge

L'efficacité du mappage d'une application parallèle sur les grappes et les grilles d'ordinateurs constitue un vrai challenge. Une des principales caractéristiques de ces plates-formes est l'hétérogénéité des nœuds qui les composent. Par ailleurs, les nœuds peuvent disparaître à cause d'une panne ou de l'utilisation du nœud par un autre utilisateur. Dans le cas spécifique des grilles, des nouveaux nœuds peuvent même apparaître en cours d'exécution. La charge au sein de ces plates-formes peut montrer également, des variations très dynamiques et souvent imprévisibles. La charge d'un nœud correspond à la quantité de travail accomplie par son système, durant une période donnée. Elle peut être représentée par de nombreuses manières différentes. Par exemple, elle peut être quantifiée par le nombre de processus en train d'utiliser ou d'attendre la disponibilité d'un processeur ou d'une ressource externes (données ou paramètres).

La charge se caractérise au moyen de fonctions et de paramètres dont l'objectif est de construire un modèle capable de capturer, de montrer et de reproduire celle-ci, ainsi que ses caractéristiques les plus importantes. Elle peut être ainsi modélisée de différentes manières

selon le domaine d'application. Par exemple, les applications internet le seront plutôt en termes de charge multimédia de type transfert d'images et de fichiers audio. Dans le domaine qui nous concerne, c'est-à-dire le calcul distribué, la caractérisation porte en particulier, sur la charge CPU et sur les entrées/sorties de tous les nœuds de la plate-forme. Chacune de ces charges est décrite par un ensemble de paramètres. Certains de ces paramètres sont statiques, quand ils sont par exemple relatifs à l'utilisation des ressources matérielles ou logicielles. D'autres sont dynamiques s'ils ont trait au comportement de l'application. La mesure de ces paramètres donne une indication sur l'état de la charge et des traitements en cours sur chaque nœud et donc de la plate-forme toute entière. L'approche communément adoptée pour la caractérisation de la charge globale est de type expérimental. Elle est basée sur l'analyse des mesures collectées sur l'ensemble de la plate-forme lors du calcul de charge. Du fait de la complexité de celle-ci, l'évaluation de sa charge devient très vite un vrai challenge.

La nature dynamique de la charge est à l'origine de problèmes spécifiques sur les performances de l'application et sur leur évaluation. Voici les questions que nous pouvons nous poser : quels sont les scénarios sur lesquels s'appuie l'évaluation des performances et quels types d'objectifs sont attendus ? Les approches adéquates doivent souvent combiner des métriques de performance appropriées, des modèles réalistes de charge et des outils flexibles pour générer, soumettre et analyser la charge, sans être pénalisants en terme d'évaluation. Il apparaît clairement que ni un système d'évaluation a priori de la charge, ni l'analyse statique de cette dernière, ne sont capables de répondre à eux seuls à tous ces objectifs.

Dans le premier cas, la simulation des évaluations des performances, le système ne prend en compte qu'un nombre restreint de paramètres fixés au départ (nombre et puissances de nœuds, etc.), qui ne permet pas une analyse de l'évolution de la plate-forme à plus long terme. Les systèmes simulés sont de cette façon restreints aux éléments que le programmeur a envisagés initialement et les résultats de la simulation ne reflètent pas la réalité, mais plutôt une certaine tendance du comportement dans le temps, de la variation de la charge. De plus, ces évaluations produisent des résultats difficilement reproductibles.

Dans le deuxième cas, l'utilisation d'un mappage ou d'un équilibrage de la charge de type statique, présente des problèmes similaires à ceux évoqués précédemment. Les informations sur lesquelles s'appuie le système pour effectuer toutes ses opérations et ses calculs, sont connues à l'avance, avant le lancement même de l'application concernée. Les différentes tâches de l'application sont donc allouées durant la compilation, sur les différents nœuds de la plate-forme, en accord avec cette connaissance a priori qu'a le programmeur de cette dernière. Cette allocation sera difficilement modifiable en fonction de l'évolution du support d'exécution. L'efficacité de l'exécution de l'application ne peut donc être garantie si la plate-forme connaît des variations importantes dans sa composition. En effet, l'accent est trop mis sur les variations dans la composition de la plate-forme et pas assez sur les variations de charge induite par les autres utilisateurs. Ce qui entraine la nécessité d'introduire des mécanismes d'arrêt et de redémarrage des nœuds de la plate-forme et empêche ces systèmes d'être tolérants aux pannes et aux fautes. La solution d'utiliser une plate-forme auto-adaptive et dynamique, que nous proposons, évite ce genre d'inconvénients.

2.3.2 Nécessité d'un équilibrage dynamique de la charge

Ne pouvant se reposer entièrement sur un équilibrage statique, ni sur les simulations du système, il faut des systèmes qui implémentent et offrent des mécanismes leur permettant de s'auto-adapter à chaque instant. Plusieurs facteurs contribuent à la variation de l'état général de la charge d'une plate-forme d'ordinateurs durant l'exécution d'une application :

- L'application peut avoir elle-même un comportement imprévisible en terme de charge de calcul des différentes parties qui la composent.
- L'autonomie des nœuds, qui apporte un asynchronisme dans le fonctionnement de l'ordonnancement et dans le comportement des communications, avec des temps de latence variables (variations des performances du réseau dues aux erreurs de transmissions et à son engorgement) : grappes et grilles d'ordinateurs sont des systèmes faiblement couplés.
- Les performances hétérogènes des nœuds qui sont dues à des configurations très différentes : vitesse des processeurs, capacités mémoires.
- La variabilité de la charge des nœuds : le partage des ressources avec d'autres utilisateurs et applications, et l'utilisation des processus de l'application, en constituent les causes majeures.
- L'environnement d'exécution peut être dynamique : des nœuds peuvent disparaître ou apparaître. Cette variation dans le nombre de nœuds de la plate-forme peut être causée par le retrait volontaire des machines de certains utilisateurs, ou par des pannes survenant de manière aléatoire et indépendante, ainsi que leurs réparations.

Tous les cas évoqués ci-dessus montrent que la sensibilité des performances d'une application distribuée est une conséquence directe de la complexité de l'interaction entre l'application et les ressources de la plate-forme. D'une part, le management et l'allocation des ressources doivent évoluer en fonction de l'état d'avancement de l'exécution de l'application. Et d'autre part, l'application doit se montrer très flexible et résiliente aux changements dans la constitution et dans la disponibilité des ressources de la plate-forme. Ni les solutions spécifiques développées par le programmeur, ni un équilibrage statique de la charge ne peuvent répondre efficacement et de manière définitive, à ces problèmes. Le programmeur peut bien évidemment prendre en compte tous ces aspects dans le développement d'une application distribuée. Mais, outre le temps que pourraient lui prendre les développements supplémentaires pour ajouter cette gestion de la flexibilité, il créera du code non portable et ne pourra exécuter son application que dans un environnement spécifique.

L'utilisation d'algorithmes d'équilibrage montre également des limites dans l'efficacité de la prise en compte de la complexité de l'interaction entre l'application et les ressources de la plate-forme. Les solutions proposées dans ce cadre par les algorithmes traditionnels, sont en effet, souvent statiques. Elles ne prennent pas en compte le caractère imprévisible de la plate-forme d'exécution. Nous avons donc mis en œuvre une solution plus dynamique, qui réordonnance les tâches en cours d'exécution et change la granularité des calculs. Ce réordonnancement se base sur des prédictions de la charge, qui sont calculées à partir de données de charges passées.

2.3.3 Prédiction de la variation de la charge

Dans un environnement partagé, les applications sont en concurrence active avec des charges inconnues de travail, qui sont introduites sur chaque nœud, par d'autres utilisateurs. C'est ce partage des ressources qui amène une grande variabilité de la charge, dans le temps. Observer le comportement d'une application lors de son exécution n'est pas suffisant pour obtenir les meilleures performances possibles. Il faut anticiper les changements dans la charge et dans la constitution des nœuds de la plate-forme d'exécution, en s'appuyant sur des mesures effectuées dans un passé proche.

De nombreuses études prouvent que les différents niveaux de charge d'une CPU sont fortement corrélés dans le temps. Ce qui rend possible des schémas de prédiction basés sur des historiques de la charge. Pour être efficaces, ces schémas doivent modéliser les relations de ces données relevées dans l'historique avec les valeurs futures de la charge. Beaucoup de modèles analytiques proposent des solutions un peu trop statiques pour traiter des variations de charge impromptues. Ils tentent d'estimer la probabilité d'obtenir la CPU ou de calculer sa charge moyenne. En fait, ces deux méthodes sont orthogonales. Il s'agit dans les deux cas, de calibrer la quantité de CPU et les intervalles ou quanta de temps qui pourraient être alloués à l'exécution d'un processus. Un problème surgit, lors de cette évaluation : si la moitié du temps de la CPU est disponible, un processus devrait être en mesure d'espérer pouvoir disposer de l'autre moitié. Il a été prouvé que ceci n'est cependant pas le reflet de la réalité. Le pourcentage disponible de la CPU agit comme un facteur d'expansion pour déterminer le temps potentiel d'exécution d'un processus. Ainsi par exemple, si seulement la moitié des quanta de temps est disponible, l'exécution d'un processus prendra au moins deux fois plus de temps que s'il était exécuté seul sur le processeur (sans compter les délais de changement de contextes des processus (piles et variables d'état).

Un bon modèle de prédiction doit donc tenir compte de la puissance et de la politique d'ordonnancement des processus des unités de calcul (CPU) de chaque nœud d'une grappe de calcul. Les modèles de prédiction sont souvent basés sur un modèle d'exécution de type Round Robin. Ils considèrent que la CPU est également partagée entre tous les processus se trouvant localisés dans une file d'attente. Ainsi, la charge du système peut être caractérisée à chaque instant, par le nombre de processus qui sont présents dans la file d'attente. La probabilité d'obtenir la CPU est dans ces conditions, inversement proportionnelle à ce nombre. Malheureusement, ces modèles caractérisent seulement des processus liés à la CPU. De tels processus sont toujours prêts à être exécutés par le processeur. Ainsi, ils peuvent consommer entièrement tous les intervalles ou quanta de temps qui leur sont alloués. Ils tirent parfaitement bénéfice du temps de la CPU disponible. Le partage entre les processus de la CPU, a lieu de manière équilibrée, selon le principe donné par le modèle Round Robin. Le problème arrive lorsque se trouvent dans la file d'attente, des processus liés aux entrées/sorties. Ce type de processus ne peut pas tirer entièrement avantage des temps CPU qui lui sont accordés, soit parce qu'il n'est pas encore prêt à être exécuté, soit parce qu'il n'a pas consommé tous les quanta de temps alloués. Ce dernier type de processus rend plus difficile la prévision sur la disponibilité et la charge des nœuds dans une grappe de calcul. Ainsi par exemple, si deux nœuds ont chacun un processus à exécuter dans leur file d'attente, le modèle Round Robin prévoit les mêmes conditions d'attribution de la CPU, pour tout nouvel arrivant sur chacun de ces nœuds. Mais si les processus sont de nature différente (l'un est de type lié à la CPU et l'autre lié à des entrées/sorties), les temps d'attribution de la CPU seront fortement différents. Ceci montre bien qu'il faut des modèles ou des manières de procéder plus ou moins complexes, pour décrire et anticiper le partage de la CPU.

Nous venons donc de décrire les écueils qui peuvent venir entraver le bon fonctionnement des systèmes de prédictions. Nous proposons un modèle de prédiction de la charge des nœuds de calcul, qui permet d'éviter ce genre de problème, et accroît les capacités auto-adaptatives de l'application. Avec notre solution, l'application peut se montrer réactive et même proactive à tout changement de la configuration et de la charge du support d'exécution, ainsi qu'à ceux provoqués par sa propre exécution.

2.4 Impact de la variation de la charge sur l'équilibrage

2.4.1 Gestion de la charge au niveau de l'application

La plupart des changements qui peuvent influer directement sur les temps d'exécution d'une application distribuée sur une plate-forme de type grappe de calcul, proviennent en particulier des variations de la charge et de la disponibilité des nœuds. L'équilibrage dynamique de la charge est donc essentiel, si l'on veut tirer avantage des ressources disponibles de la plate-forme de calcul et obtenir ainsi des performances optimales. Il est nécessaire de savoir bien gérer cette variabilité pour pouvoir obtenir des algorithmes d'équilibrage efficaces, et qui puissent être utilisés à toute échelle.

Cet équilibrage dynamique peut bien sûr être réalisé au niveau de l'application elle-même. Dans ce cas, cette dernière doit continuellement mesurer et détecter les déséquilibres de la charge et essayer de les corriger en redistribuant les données, ou en changeant la granularité des calculs par la redistribution et la migration de ses composants sur la plate-forme. Une telle approche connaît cependant plusieurs limites assez rédhibitoires pour être utilisée dans n'importe quelle grappe de calcul. En premier lieu, elle demande une programmation complexe et des connaissances spécifiques pour pouvoir aborder ce domaine de l'équilibrage de la charge. Cette approche n'est donc pas vraiment transparente pour le programmeur. Deuxièmement, si ce type d'applications autorégulées peut parfaitement bien fonctionner sur certaines grappes de calcul aux caractéristiques prédéfinies, leur fonctionnement est plus problématique lors de l'utilisation de plates-formes dont la disponibilité et l'utilisation des ressources demeurent plus fluctuantes. La même conclusion s'impose que pour le mappage d'une application sur les grappes de calcul : il est préférable que l'équilibrage de la charge et la gestion des ressources soient effectués par un système tout à fait à part, de type middleware (intergiciel).

2.4.2 Gestion de la charge par un intergiciel

Dans ce type de système où la régulation de la charge est gérée par un intergiciel, toute décision concernant l'équilibrage doit tenir compte des informations rassemblées dans un système global ou même local, à partir des données sur l'état de la charge de chaque nœud. Si l'état du système subit des petites variations, les informations peuvent être mises à jour facilement. Par contre, si ces variations sont très importantes, les informations risquent de devenir obsolètes pendant la durée des décisions sur l'équilibrage. De ce fait, la probabilité de prendre une mauvaise décision sur la répartition des charges augmente considérablement.

Pour maitriser les effets de la variabilité de la charge sur les performances de l'équilibrage, nous proposons de décomposer les algorithmes d'équilibrage de charge en quatre étapes :

- Mesure locale de la charge : la décision d'équilibrage de charge repose toujours sur une mesure de la quantité de charge et/ou d'une information de disponibilité des nœuds. Cette information est couramment quantifiée à l'aide d'un index spécifique relatif à la charge. La non connaissance a priori, des effets des index et des métriques utilisés, peut donner la tentation d'instrumenter le code à outrance, afin d'obtenir les meilleures performances possibles. Une instrumentation irraisonnée du code peut générer des gros volumes de données de performance. Elle perturbe ainsi le comportement de l'exécution de l'application et nuit à son bon déroulement. Les instruments d'observation mis en place doivent être aussi légers, et avoir le minimum d'intrusion, que possible. Ils ne doivent être utilisés que lorsqu'on en a vraiment besoin et ne doivent récolter que le minimum de données requis.

- Règle d'information : la politique, ou règle d'information, spécifie la manière d'échanger entre les nœuds les informations sur leur état. La politique d'information peut être aussi bien locale que globale, suivant les cas. Selon le caractère qualitatif et quantitatif des données récoltées dans la précédente étape, il vaut mieux prendre certaines décisions localement et ne pas échanger toutes les données. En effet, comme les données sur la performance des nœuds et de l'application sont capturées localement, les décisions prises à distance peuvent ne pas concorder avec les contraintes temporelles de l'exécution locale et être obsolètes lorsqu'elles arrivent. Dans d'autres cas, il y a des décisions globales à prendre et il vaut mieux échanger les informations sur les données de performance, de manière systématique. Ce sont par exemple, les cas où les décisions de rééquilibrage de la charge nécessitent une connaissance globale de l'état de la plate-forme. Il faut donc autant que possible, analyser et traiter localement sur les lieux de capture, les données relatives aux informations de performance (charges et disponibilités des nœuds) et véhiculer le moins possible celles-ci.

- Règle d'initialisation : cette étape détermine le moment où il faut initier les opérations d'équilibrage de charge. Comme ces opérations introduisent un surcoût temporel (overhead) non négligeable, il faut veiller à ce que ce coût ne soit pas supérieur à celui des bénéfices obtenus par les opérations d'équilibrage de charge.

- Opération d'équilibrage de charge : cette dernière opération peut se décomposer en trois sous-étapes : les étapes d'éligibilité, de distribution et de sélection. L'étape d'éligibilité détermine le ou les meilleurs nœuds candidats pour participer aux opérations d'équilibrage de charge. La règle de distribution détermine la manière d'équilibrer la charge entre les nœuds impliqués dans cette opération. Cet équilibrage peut se faire par des exécutions à distance (allocation ou placement initial des tâches non préemptif) ou par des migrations de processus. Enfin, la règle de sélection choisit la partie de la charge qui doit être distribuée ou redistribuée lors de cette opération.

Pour mettre en œuvre ces quatre étapes nous proposons d'instrumenter le code de l'application et d'analyser à la volée les données remontées par les instruments d'observation. Un mécanisme de contrôle exploite ces analyses et reconfigure dynamiquement l'application en fonction des variations dans son exécution et de celles du support d'exécution. C'est de cette manière que fonctionne l'intergiciel ADAJ que nous avons développé. Ses différents

mécanismes s'adaptent à tous types de plates-formes de calcul et de configurations matérielles et logicielles. L'amélioration dans les performances de l'exécution de l'application, est ainsi obtenue par le meilleur choix possible dans les configurations des ressources, en fonction des variations temporelles du comportement de l'application. Les performances sont liées aux différents algorithmes mis en œuvre dans chacune des quatre étapes.

2.4.3 Analyse des effets de la variation de la charge

Les variations de la charge peuvent être plus ou moins importantes. Nous ne tenons pas compte des faibles variations de charge, qui ne perturbent pas en principe, l'équilibre général de la charge sur une plate-forme d'exécution. Par contre, l'analyse des effets d'une variation importante de la charge a orienté nos travaux. Elle porte sur chacune des quatre étapes énoncées dans le paragraphe précédent (mesure de la charge, spécification de la règle d'information, spécification de la règle d'initialisation et opération d'équilibrage de la charge) et impose les conclusions suivantes :

Les effets d'une forte variation de la charge sur la première étape reposent sur l'index de charge que l'on a choisi. Un bon index est capable de prendre en compte ces variations dans ses calculs pour en préserver la mémoire et maintenir une certaine stabilité de l'état des nœuds. Ainsi l'efficacité de cette étape dépend entièrement de l'index de charge choisi. Un bon choix permettra de bien gérer la variabilité de la charge.

L'impact de la variabilité sur la politique d'information est également très important. Plus la variabilité de la charge sera forte, plus il sera difficile de maintenir à jour les informations sur l'état des nœuds de toute la plate-forme, sans rajouter un overhead important dû aux transferts sur le réseau. Il est donc nécessaire de disposer du meilleur mappage initial possible des tâches de l'application.

Concernant la politique d'initialisation de la troisième étape, les effets de la variabilité sont directement liés à la politique d'information adoptée lors de la seconde étape. Si elle est capable de gérer convenablement la variabilité de la charge et de maintenir correctement des informations sur les états de tous les nœuds, les décisions d'initiation seront prises de manière satisfaisante et les variations n'affecteront pas cette étape.

L'efficacité des opérations d'équilibrage de charge peut être affectée par les variations de celle-ci, dans les étapes d'éligibilité et de distribution. La politique d'éligibilité sélectionne le meilleur candidat pour participer à ces opérations. Même si le système d'information participant à ce choix est à jour, il reste encore un problème. Dans un environnement où la charge varie de façon intensive, l'état du système peut changer avant même que le choix du nœud cible ne se fasse. Le meilleur candidat pour un instant donné peut se révéler être le plus mauvais immédiatement après. La politique de distribution peut être basée soit sur l'exécution à distance, soit sur la migration de processus. Chacun des ces deux modes de fonctionnement présente des avantages et des inconvénients. Dans certains cas, la migration ne fournit pas de meilleurs résultats que l'exécution des tâches et processus à distance, à cause des problèmes de coûts de transfert. D'un autre côté, l'utilisation de la migration peut apporter des améliorations significatives dans les performances, notamment lorsque des systèmes instables du point de vue de la charge sont utilisés. Une tâche peut être allouée à un nœud donné dont

l'état de la charge se révèle par la suite un peu trop instable. Dans ce cas, il est préférable de réallouer les tâches et de faire migrer le processus pour gagner en performance.

Les effets d'une variation importante de la charge sur les différentes étapes dans l'équilibrage de celle-ci, nous ont amenés à faire face à plusieurs défis. Ceux-ci sont relatifs notamment au choix de l'index de la charge, à la conception du système d'information et à l'exécution des opérations d'équilibrage de charge.

2.5 Equilibrage de la charge par la migration

2.5.1 Bénéfices attendus

La migration de processus, ou d'objets, permet de distribuer dynamiquement la charge entre des nœuds surchargés et d'autres qui le sont moins. Mais elle apporte également des bénéfices supplémentaires. Par exemple, elle permet une meilleure gestion de la plate-forme d'exécution, en facilitant la gestion de la tolérance aux pannes, l'accession locale aux données et une réaction aux changements de la plate-forme d'exécution :

- La tolérance aux pannes : si le nœud sur lequel s'exécute le processus tombe partiellement en panne, sa migration vers une machine plus sûre permet de continuer l'exécution sans problème.

- L'accession locale des données : la migration permet d'acheminer le processus vers les sources de données sur lesquelles il travaille. Elle permet de transformer des accès à distance en communications locales. Elle évite ainsi des surcoûts en communications qui nuisent aux performances globales de l'application.

- L'amélioration de l'administration du système avec une meilleure réaction aux changements dans la constitution de la plate-forme d'exécution : s'il est possible de détecter que des nœuds vont s'éteindre électriquement et se déconnecter du réseau dans un futur proche, la migration de processus permet d'anticiper cet arrêt et de poursuivre l'exécution de l'application, en la redéployant sur d'autres nœuds.

2.5.2 Migration des processus

Un processus est une abstraction dérivant des systèmes d'exploitation qui représente une instance d'un programme en cours d'exécution. Ce concept a été introduit en 1992, par Tannenbaum. Un processus regroupe des informations sur l'état courant de l'exécution. Il rassemble donc des données sur l'état de la pile d'exécution, le contenu des registres, ainsi que d'autres paramètres spécifiques au système d'exploitation sous-jacent, qui concernent le processus lui-même, la mémoire et le système de gestion des fichiers. Un processus peut être composé d'un ou de plusieurs threads. Les threads appelés aussi processus légers possèdent leur propre pile et registres d'exécution, mais partagent un espace commun d'adressage avec

d'autres threads du processus et des signaux (concept spécifique aux systèmes d'exploitation). Le concept de tâche a été introduit comme une généralisation du concept de processus, cependant un processus est décomposé en une tâche et un certain nombre de threads. Classiquement, un processus est représenté par une tâche et un processus de contrôle.

La migration de processus est l'acte de transférer durant l'exécution de ses threads, un processus entre une machine source et une machine cible. D'un point de vue de plus haut niveau, elle consiste à extraire l'état du processus du nœud source, de le transférer à destination d'un nouveau nœud où une nouvelle instance du processus est créée, et à mettre à jour toutes les connexions avec les autres processus communiquant avec lui. La migration de tâche consiste au transfert d'une tâche pendant l'exécution des threads qui la composent. Pendant le transfert, deux instances de la tâche coexistent : l'instance « source » qui est la tâche initiale et l'instance « destination » qui est la nouvelle tâche fraîchement créée sur le nœud destination. Si l'on a défini un nœud hôte sur lequel la tâche est liée logiquement de manière privilégiée, cette tâche qui tourne après migration sur le nœud destination est vue comme une tâche distante (remote en anglais). Dans ces conditions, une invocation à distance (remote invocation) est la création d'une tâche sur un nœud distant. Cette opération est moins coûteuse qu'une vraie migration. Elle n'implique pas la migration de tout le contexte d'exécution, mais seulement le code, certains états et les fichiers ouverts. Cette migration peut se faire également sans les données et ne concerner que le seul code, comme dans le cas des applets Java. Elle peut également concerner le code, les données et certains paramètres comme les droits d'accès aux données. Ce type de migration est réalisé par des systèmes à agents mobiles.

Il peut exister quelques inconvénients à utiliser la migration pour équilibrer la charge. Les systèmes de migration mettent en œuvre des politiques différentes pour la suspension et la reprise de l'application. Si les systèmes au départ et à l'arrivée du processus sont différents, le temps entre la suspension et la reprise du processus sur le nouveau système peut être assez long. D'autre part, l'utilisation de conditions prédéfinies pour la suspension et la migration, et l'absence de connaissance sur le temps qu'il reste pour exécuter le processus sur l'ancien système, peuvent entraîner une suspension et une migration du processus, même s'il ne lui restait que peu de temps pour finir son exécution.

2.5.3 Migration en Java

Un langage objet comme Java utilise tous les types de mécanismes que nous venons de décrire pour la migration de processus. La migration d'objets s'exécutant dans une machine virtuelle (JVM) est donc semblable à la migration de processus, si l'objet qu'il faut migrer possède au moins une méthode en cours d'exécution dans un thread. Dans ces conditions migrer un objet revient fonctionnellement à migrer le thread dans lequel s'exécute la méthode. La migration d'un objet d'une JVM à l'autre, est donc réalisée par un transfert physique. Toute nouvelle invocation faite sur l'objet qui a été migré doit se faire sur le nouvel emplacement. La migration d'objets s'inspire donc du modèle de programmation à processus et propose différentes stratégies. La migration peut être soit de type faible, soit de type fort. La migration faible suppose de redémarrer après le transfert en recommençant sur le nœud destinataire, depuis le début, l'exécution interrompue. La cohérence de l'état de l'objet ne peut être garantie que s'il n'existe pas de méthode s'exécutant sur l'objet. Dans le cas contraire, le risque est d'avoir une double exécution des instructions des méthodes

concernées. En tenant compte de l'état courant de l'exécution, la migration forte élimine ce risque.

Les différentes techniques de migration reposent sur le transfert du contexte d'exécution : programme, données et état d'exécution. L'environnement Java permet la migration de code par le téléchargement dynamique du bytecode par les JVM et la migration des données par le mécanisme de sérialisation. En revanche, la migration de threads n'est pas totalement possible dans cet environnement, à cause de l'impossibilité de migrer également l'état d'exécution. Cependant l'information nécessaire à cette opération se trouve dans la machine virtuelle, mais les mécanismes de sécurité de Java interdisent de récupérer cette information. Trois solutions existent pour remédier à ce problème : étendre en la modifiant la machine virtuelle, étendre le langage Java à l'aide d'un préprocesseur, ou étendre le compilateur par transformation du bytecode :

- L'extension de la machine virtuelle implique que les applications qui souhaiteraient faire appel aux mécanismes de migration, s'exécutent sur des JVM non standards qui ont été modifiées. L'inconvénient de cette approche est donc la non portabilité des applications développées dans ce contexte.
- L'extension du langage Java ne permet pas quant à elle, d'extraire entièrement l'état d'exécution d'un thread, comme par exemple les valeurs sur la pile d'opérande. L'efficacité d'une telle technique peut être limitée par l'utilisation d'instructions conditionnelles imbriquées, qui sont mises en œuvre pour simuler des instructions de transfert de contrôle (typiquement l'instruction goto) qui permettent d'éviter la reprise d'un code déjà exécuté.
- L'extension du compilateur, au niveau de la post-compilation, se montre plus efficace que l'utilisation de techniques d'extension du langage comme décrites précédemment. Un des avantages de la transformation du bytecode repose sur la possibilité d'insérer des instructions de transfert de contrôle (typiquement l'instruction goto) dans le bytecode. L'introduction de ce type d'opération demanderait au langage Java d'augmenter considérablement le bytecode.

2.5.4 Problèmes posés par la migration

Réalisé dans un environnement partagé et dynamique, l'équilibrage de la charge par la migration de processus ou de threads rencontre cependant quelques limites apportées par la granularité de l'application. Comme dans un tel environnement, il est impossible de prévoir précisément quelle sera la disponibilité des ressources au lancement de l'application, il est par conséquent difficile de déterminer quelle est la bonne granularité initiale à adopter pour son exécution.

Adopter une granularité fine en surestimant le nombre de ressources disponibles de la plate-forme, peut conduire à des performances dégradées, dans le cas d'une famine dans les ressources. A l'opposé, adopter une grosse granularité peut conduire l'application à ne pas utiliser toutes les ressources potentielles de la plate-forme et ainsi nuire à son efficacité générale. Pour corriger ce problème, il y a d'abord la solution de mise en œuvre des techniques d'équilibrage statique de la charge, qui permet d'avoir un placement initial optimal des différentes parties de l'application sur la plate-forme. Cependant si les variations

de la plate-forme sont trop importantes, il faut ensuite mettre en œuvre un équilibrage de la charge de type dynamique.

Ainsi, quand seule la migration de processus est mise en œuvre, l'équilibrage de la charge est vite confronté aux limites de la granularité de l'application. La solution à ce problème, consiste à utiliser l'aptitude des processus de certaines applications à pouvoir s'épandre et se contracter selon les disponibilités des ressources de la plate-forme. De telles applications sont dites malléables. Ce caractère de malléabilité permet donc à l'application de gérer dynamiquement sa granularité en utilisant le mécanisme de migration de processus. Ce caractère est géré en même temps au niveau opérationnel et à un méta-niveau. Le niveau opérationnel détermine comment faire la décomposition et le regroupement des processus tout en préservant la sémantique et l'intégrité de l'application. Il est fortement lié au modèle de programmation et nécessite le développement de librairies spécifiques. La gestion au méta-niveau quant à elle, rend compte du moment où il faut faire les opérations de regroupement ou de décomposition, et du bon mappage de l'application sur le support d'exécution. Cette dernière opération de la malléabilité est une tâche très difficile et complexe à réaliser. Il est donc préférable que cette gestion soit, elle aussi, réalisée par un middleware capable de prendre en compte ce méta-niveau.

2.6 Etat de l'art

2.6.1 Exemples de mappage d'applications parallèles

Beaucoup d'algorithmes et de méthodes ont été proposés, pour mapper les applications parallèles sur des plates-formes hétérogènes. La plupart d'entre elles sont de type statique, mais il en existe quelques-unes de type dynamique. Comme le montrent les exemples que nous présentons ci-dessous, ces dernières utilisent des solutions approximées d'un mappage optimum. Elles diffèrent par la façon dont elles modélisent une application parallèle (DAG, TIG), par les algorithmes mis en œuvre (chemin critique, recherches locales, algorithmes évolutionnaires) et par les différents paramètres qu'elles utilisent (coût de calcul des tâches, puissance de calculs des processeurs, affinités entre les machines et les tâches).

2.6.2 Exemples d'applications auto-adaptatives

Le concept de calcul autonome (autonomous computing) regroupe les approches dont la gestion de la charge et de l'auto-adaptativité est réalisée par l'application elle-même. La plupart des solutions mises en œuvre dans ce cadre ont pour objectif de minimiser un « makespan » (temps mis par le dernier des processus s'exécutant). Ces systèmes diffèrent selon le type d'environnement envisagé, le type de logiciel adaptatif mis en place et le moment où les mécanismes de l'auto-adaptativité sont employés. Quelques exemples de systèmes auto-adaptatifs développés au niveau du logiciel, sont donnés ci-dessous :

- **ATCC (Automatically Tuned Collective Communications) [VAD00]** est utilisé pour optimiser les communications collectives de type MPI (Message Passing Interface), sur un ensemble de machines connectées par réseau avec une configuration spécifique. Les routines de communications collectives sont exploitées par bon

nombre d'applications parallèles basées sur l'utilisation de ce mode de programmation MPI. Pendant son installation, ATCC mène des expériences concernant différents algorithmes et différentes politiques de communications collectives avec, pour paramètres, le nombre de processeurs et la taille de message. ATCC regroupe ensuite les résultats de ces informations dans une table de temps passé. Quand arrive ensuite, de la part de l'utilisateur, une commande de communication collective avec une taille de messages et un nombre de processeurs donnés, ATCC n'a plus qu'à consulter cette table pour déterminer le meilleur algorithme de communication collective et la meilleure politique possible. Des versions récentes de ATCC incorporent des modèles de performance pour les algorithmes de communications collectives, afin de réduire le temps que prennent les expérimentations réelles.

- **Le projet de BeBOP** (Berkeley Benchmarking and OPtimization) **[BeBOP]** qui est mené à l'Université de Berkeley optimise les calculs sur les noyaux des matrices creuses (multiplication de matrices vecteurs, triangularisation de matrices) pour des architectures bien ciblées. Pour chacun des noyaux de matrices creuses, BeBOP prend en considération un ensemble d'implémentations et choisit une implémentation optimale ou quasi-optimale qui tient compte de l'architecture cible. Ce choix se fait en deux étapes. En premier lieu, les implémentations potentielles sont étalonnées hors connexion, de manière indépendante de la matrice, mais dépendante de la machine. Lorsque la structure de la matrice est connue à l'exécution, la matrice est échantillonnée de manière à extraire les aspects relatifs à sa structure, et aux modèles de performance qui combinent les données du benchmark. Les propriétés de la matrice estimées sont alors évaluées pour obtenir une implémentation quasi-optimale.

- **Le projet LFC (LAPACK for Clusters) [VAD00]** vise à simplifier l'utilisation des logiciels parallèles d'algèbre linéaire, sur grappe de calculs. En particulier il permet de résoudre des systèmes linéaires pour des matrices denses, sur des systèmes parallèles. Des benchmarks sont réalisés sur des noyaux séquentiels qui sont invoqués par des logiciels parallèles. L'auto-adaptativité est réalisée par des algorithmes d'ordonnancement pendant l'exécution. Ceux-ci prennent en compte les caractéristiques de l'environnement d'exécution et choisissent de façon optimale ou quasi-optimale, un sous-ensemble de ressources pour l'exécution de l'application parallèle. LFC optimise aussi les paramètres du problème (taille du bloc de la matrice). Il autorise les invocations à distance du logiciel parallèle, par un environnement séquentiel, et met en œuvre pour cela des stratégies, comme par exemple, les déplacements des données. Cette approche a permis de résoudre des problèmes de ScalaPACK LU, QR et les routines de factorisation de Choleski. Ces routines permettent la résolution de très grands systèmes denses non réalisables avec des architectures conventionnelles de calcul.

Tous les exemples que nous avons trouvés pour illustrer cette classe de calcul autonome, se rapportent des applications particulières. Leurs mises en œuvre nécessitent pour la plupart d'entre elles, la formation des développeurs à un modèle de programmation et à un moteur d'exécution spécifiques.

2.6.3 Exemples de middlewares auto-adaptatifs

La gestion de l'auto-adaptativité par un middleware a été d'abord étudiée pour des grappes de calcul. Les fonctionnalités de ces intergiciels ont été étendues dans un second temps pour pouvoir s'exécuter sur des grilles de calculs :

- **Autopilot [RIB01]** réalise une optimisation dynamique de l'exécution de l'application en intégrant une instrumentation et une politique d'exploitation des ressources et des procédures de décision. Des capteurs distribués rassemblent des données quantitatives sur l'application et sur les performances des systèmes. Des entités logicielles de reconfiguration (actuators) exploitent ces informations et rendent possibles ensuite, les redéploiements dynamiques de l'application en fonction des politiques spécifiques de gestion de ressources. Tous ces éléments sont inclus dans le code source d'application. Les entités logicielles de reconfiguration sont contrôlées pendant l'exécution par des procédures de décision qui sont basées sur des techniques de logique floue. Pour décrire des schémas de comportement sûrs, Autopilot met également en œuvre des outils de classification de comportements. Ces outils sont basés sur les techniques des chaînes Markov et des réseaux neuronaux. La faisabilité de cette approche a été testée dans le projet PPFSII (Portable Parallel File Système) [VET00]. Les directives données par le programmeur sont vérifiées en ligne pendant qu'il écrit son code.

- **DIET (Distributed Interactive Engineering toolbox [DIE11]** est le travail conjoint de plusieurs équipes INRIA, qui a été démarré en 2000. DIET est une plate-forme de métacalculs et multiagents qui a été implémentée à l'aide du paradigme de communication CORBA. Chaque agent interconnecté avec les autres, gère une partie de la plate-forme de calcul. Cette gestion est accomplie de manière hiérarchique, sans aucune centralisation. DIET repose en partie sur le composant FAST (Fast Agent's System Timer) afin de détecter et d'instrumenter les ressources disponibles sur chaque nœud de la plate-forme, de façon dynamique. FAST permet aussi de construire une métrique de capacité de chaque nœud basée sur un jeu de tests. Cet étalonnage permet ensuite d'évaluer le temps nécessaire à l'exécution d'une application sur un nœud donné, et donc de prévoir l'utilisation future du nœud.

- **Javelin 3** [**NEA02]** est une infrastructure qui a été développée en Java. Elle est composée de clients qui recherchent des ressources, d'hôtes qui offrent des ressources et de courtiers qui coordonnent les allocations. Outre cette architecture réalisant un ordonnancement dynamique des tâches, Javelin implémente un modèle de programmation de type Branch and Bound. Les tâches parallèles sont décomposées en ensembles de sous-tâches autonomes du point de vue des communications, puis distribuées par l'infrastructure. Le modèle Branch and Bound apporte la dynamicité au moyen d'un processus de vol de travail et d'un ordonnanceur distribué. La tolérance aux pannes est aussi introduite par un mécanisme de remplacements des nœuds tombés en panne, ou qui se sont retirés. Le modèle d'ordonnancement proposé par Javelin est donc décentralisé, avec une politique orientée vers l'application.

- **L'architecture SALSA/IO [SZY04]** comporte un langage orienté acteur (SALSA : Simple Actor Language, System and Architecture), un environnement d'exécution distribué et une infrastructure de type middleware (IO : Internet Operating System) qui équilibre la charge. Le langage SALSA gère des constructions de haut niveau pour le passage de message, un nommage universel, la migration et la coordination. Un acteur est une entité qui encapsule à la fois un état des ressources (codes et données) et le traitement de cet état (thread de contrôle). Toutes les communications entre les différents acteurs passent par des messages. Après le traitement de message, l'acteur peut réagir de trois manières : modifier son état, créer de nouveaux acteurs ou envoyer des messages aux acteurs qui sont reliés à lui (liaison pair à pair). L'architecture SALSA/IO se montre très dynamique. Elle supporte le profiling de ressources, la migration des acteurs vers un environnement optimal d'exécution ainsi que l'ajout et le retrait des nœuds de calcul. SALSA/MPI étend les capacités de SALSA au standard MPI qui permet aux processus MPI de s'exécuter sur des environnements plus conséquents, tels que les grilles de calcul. Cette dernière version de SALSA fonctionne au-dessus de n'importe quelle implémentation commerciale de MPI et utilise les connexions TCP entre les machines. Les mécanismes de SALSA prévoient d'associer un proxy acteur qui agit comme un agent d'étude de profil dans le réseau. Chaque proxy acteur de SALSA/MPI est composé d'un acteur qui étudie les profils et d'un agent. L'acteur surveille et enregistre les communications des processus MPI et les ressources matérielles (mémoire, CPU et réseaux). Les résultats de cette surveillance (monitoring) sont régulièrement adressés à l'agent de décision. S'il y a plus ou moins de ressources disponibles pour l'exécution des calculs, cet agent essaie de reconfigurer l'application en cours d'exécution, en faisant migrer un ou plusieurs processus. Cette architecture supporte la migration de processus au sein d'une même grappe de calcul, mais aussi entre différentes grappes. Cependant, il est très coûteux de disséminer les processus différents MPI d'une même application, sur différentes grappes, car ceux-ci sont souvent fortement couplés et communiquent beaucoup entre eux. Les concepteurs de SALSA conseillent donc de préférence, d'exécuter une application dans une seule et unique grappe et de n'utiliser la migration de l'application qu'en cas de panne ou en cas de grosse surcharge de la grappe.

2.7 Résumé de nos contributions

Par le projet ADAJ (références de [6], [7] et [19] à [23]), nous proposons une solution aux problèmes de l'auto-adaptativité et de l'exécution efficace des applications distribuées irrégulières en Java, sur grappes d'ordinateurs. ADAJ intègre, de façon cohérente, des outils nécessaires à une expression simple du parallélisme, et à une prise en charge automatique de la régulation de l'exécution des traitements distribués. Cette intégration se situe aussi bien au niveau application, grâce à une orientation de la méthodologie de programmation, qu'au niveau intergiciel, grâce à un mécanisme d'équilibrage de charge dynamique.

En réponse aux évolutions des calculs et aux modifications de la disponibilité des ressources, ADAJ assure une adaptation automatique de l'exécution de l'application. Il mesure dynamiquement l'état des relations entre les objets de l'application et calcule l'information relative à l'évolution de la charge des différents nœuds de la plate-forme d'exécution. Le mécanisme d'observation des relations dynamiques entre les objets fournit une connaissance

du comportement des applications pendant leur exécution. Il permet de prédire les tendances des communications entre ces objets. Il est implanté à l'aide de techniques de type post-compilation. Le mécanisme de mesure de la charge des machines, permet quant à lui, d'avoir une vue globale de la charge dynamique de chaque machine participant à la plate-forme. Il permet ainsi de détecter un déséquilibre de cette charge. L'entière conception en Java de ces deux mécanismes assure une complète portabilité du code sur l'ensemble de la plate-forme. Les systèmes d'observation transmettent les informations nécessaires au système de redistribution des objets de l'application. La plate-forme d'exécution ADAJ assure ainsi l'auto-adaptativité des applications en Java. Ces travaux ont été essentiellement menés dans le cadre des thèses d'Amer Bouchi [BOU03] et de Violette Felea [FEL03].

Les travaux précédents ont été ensuite étendus par d'autres recherches menées, dans le cadre d'un programme de coopération entre le CNRS et l'Académie des Sciences de Pologne (PICS) [PICS11]. En complément des possibilités offertes par ADAJ pour la gestion auto-adaptative de l'exécution des applications distribuées, nous avons également introduit des algorithmes d'optimisation, pour déterminer un mappage initial des objets de l'application, sur la plate-forme d'exécution. Une telle approche est fondée sur le fait que le mécanisme de l'équilibrage de charge dynamique pourra mieux fonctionner s'il est supporté par un placement initial optimisé des objets de l'application. Cette phase de calculs préalables permet en effet, de bien répartir les objets afin de réduire les temps d'exécution, non seulement au lancement de l'application, mais également en cours d'exécution. L'hypothèse prise est de considérer que les variations de la charge de la plate-forme se font progressivement. Exception faite des problèmes de modification dans la configuration des ressources matérielles (problèmes pris en compte par des mécanismes de sûreté de fonctionnement), l'exécution de l'application pourra ainsi se faire dans des conditions optimales. Le scénario de cet équilibrage de charge statique incorpore l'exécution de l'application sur un ensemble représentatif de données. Cette étape permet de détecter d'éventuelles propriétés intrinsèques de l'application relatives à des aspects de calculs ou de communications. Ces propriétés sont ensuite exploitées lors de déploiements ultérieurs.

Nous avons proposé et conçu dans ce cadre, deux méthodes (références [8], [9] et de [26] à [30]) qui s'appuient toutes les deux, sur l'analyse du bytecode d'une application qui a été généré par un compilateur Java. Cette analyse identifie les dépendances de contrôles et les dépendances de données entre les différentes méthodes et threads d'une application multithreadée. Elle permet de générer un graphe de flots de macro-données qui décrit les dépendances des flots de contrôle et de données dans les instructions du bytecode. Dans la première méthode, nous appliquons alors, un algorithme constitué de deux étapes :

- Dans la première étape, un algorithme de clustering étend ce graphe par des instructions de branchements conditionnels, annotées par des probabilités. Ces probabilités proviennent de traces d'exécution du programme sur un ensemble représentatif de données.

- Dans la seconde étape, ce graphe est réparti sur un réseau constitué par les JVM en utilisant un ordonnancement particulier appliqué sur les nœuds du graphe. Cet ordonnancement met en œuvre une heuristique basée sur le chemin le plus utilisé (most-often-used-path) et exploite l'information sur les chemins qui sont en mutuelle exclusion.

Cependant, le placement optimal des tâches sur une plate-forme composée de nœuds hétérogènes est un problème NPcomplet (NP-Hard). Nous avons donc proposé plus

récemment, une deuxième méthode (références [17], [18], [44] et [45]) pour effectuer le mappage initial. Cette méthode est basée sur un algorithme d'optimisation extrême (Extremal Optimization) qui est un cas particulier d'algorithme co-évolutionnaire très rapide. C'est une approche heuristique basée sur une fonction fitness locale qui est composée, elle-même, de deux sous-fonctions. L'élimination du temps d'exécution des tâches après réception des données nécessaires et le déséquilibre de charge dans l'exécution des tâches, sont utilisés comme les heuristiques d'entrée pour l'amélioration des solutions.

Chapitre 3

Passage à l'échelle des systèmes distribués pour le calcul et le traitement de données

3.1 Le calcul sur grille d'ordinateurs

3.1.1 Définition des grilles

Les grilles sont des très grands systèmes distribués, partagés. Elles constituent des plates-formes de traitement de l'information et de calculs distribués. En leur sein, sont reliés par des réseaux très rapides des éléments aussi divers que des unités de calcul (CPU et mémoires), des éléments de stockage plus ou moins complexes, des grappes d'ordinateurs ou des supercalculateurs. Cette définition suit celle donnée par I. Foster et C Kesselman en 1998 [FOS98] : « une grille de calcul est une infrastructure logicielle et matérielle qui fournit un accès fiable, cohérent, ubiquitaire (possibilité d'être connecté partout et tout le temps) et peu cher, à des capacités haut de gamme de calcul ». Cette définition a été remodelée en 2001, par ces deux auteurs et par S. Tueke, pour, y ajouter des aspects d'organisations virtuelle et multi-institutionnelle.

Les avancées de la technologie ont favorisé l'émergence des grilles. Par rapport aux grappes de calcul celles-ci ont bénéficié d'un gain de facteur d'échelle. Le nombre de nœuds interconnectés a, ainsi, fortement progressé grâce aux progrès réalisés dans les réseaux. Les nœuds des grilles sont devenus également plus puissants grâce à l'amélioration de la technologie des microprocesseurs. Selon la fameuse loi émise en 1965 par Gordon Moore, alors directeur de la R&D chez Fairchild Semiconductor (il fondera Intel en 1968), la puissance des microprocesseurs double tous les deux ans. Le nombre de transistors sur une puce électronique est par exemple, passé de mille dans les années 1970, à presque un milliard, aujourd'hui. Un fabricant de microprocesseurs comme Intel, envisage que ses futurs microprocesseurs multi-cœurs, soient en mesure de fournir une puissance de calcul de l'ordre du téra (1 trillion ou 1 000 000 000 000) instructions par seconde. Ainsi, cette conjoncture de Moore, qui n'était pas tout à fait vraie, jusqu'à présent (cf figure 1), va même maintenant, être dépassée dans les années à venir. Les grilles peuvent espérer avoir à leur disposition, une puissance de calcul phénoménale. D'autre part, la progression des technologies informatiques concernant les réseaux et les systèmes de stockage, a elle aussi, contribué à l'avènement des grilles.

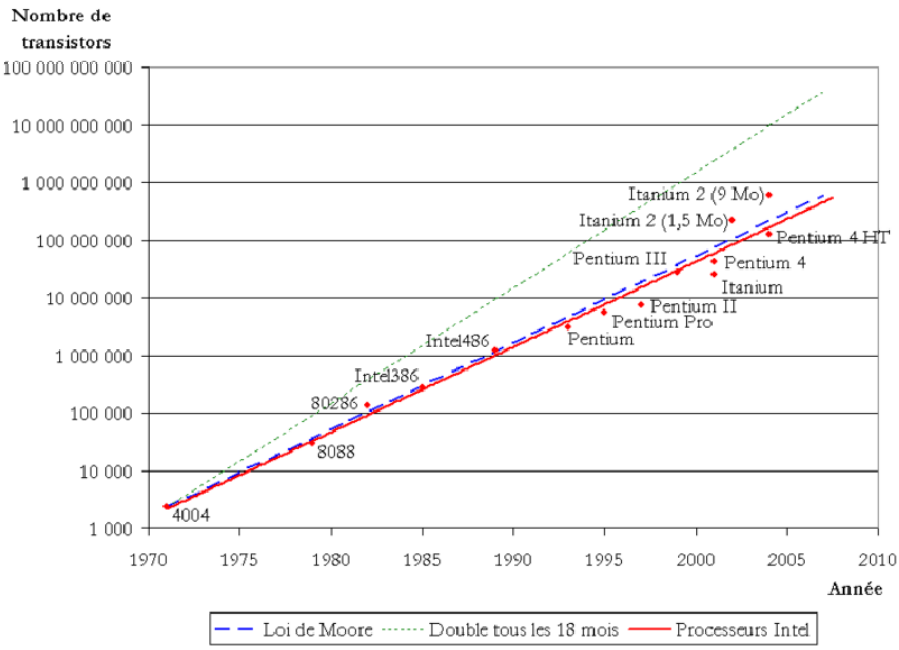

Figure 1 : Loi de Moore décrivant l'évolution des processeurs

Les performances des technologies de transmissions de données et de commutations, en particulier, ont donc beaucoup évolué. Les débits supportés par les réseaux d'ordinateurs, ne cessent d'augmenter. La courbe de progression réalisée dans le domaine des technologies réseaux suit une pente supérieure à celle réalisée dans le domaine des processeurs. Georges Gilder a énoncé une loi identique qui indique que la bande passante des réseaux augmente trois fois plus vite que la puissance des ordinateurs [GIL00]. Des calculs menés par Eric Schmidt, PDG (Chief executive officer - CEO de Google et ex-CEO de Novell), montrent que la bande passante double, en fait, tous les ans depuis 1997. Aujourd'hui, le groupe boursier NYSE Euronext s'est s'équipé d'un réseau à vitesse de transmission des informations de l'ordre de 100 Gbit/s. Le but de l'opération est de permettre à ce groupe, de réaliser plus d'un milliard de transactions quotidiennes, représentant des peta-octets de données à transférer.

Mémoire, stockage de masse, réseau : la loi de Moore, et son exigence de doubler la puissance des processeurs tous les dix-huit mois, est donc supplantée par cette autre loi de Gilder. Pour le stockage, cette loi stipule que les capacités doublent tous les neuf mois. L'évolution parallèle de ses différents composants, rend plus que jamais attractive, la technologie des grilles. Les grilles sont caractérisées par trois grands sous-ensembles :

- Les grilles de calcul : une grille de calcul est un ensemble de ressources distribuées qui est destiné à agréger les capacités de calcul de chacun des nœuds. Les nœuds de calcul peuvent être constitués de supercalculateurs. Ce type de grille est généralement mis en œuvre pour des applications haute-performance de type HPC (High Performance Computing). Ces applications comportent typiquement un ensemble de données et un espace de paramètres très volumineux dans lesquels il faut chercher. Il y a peu de

transferts de données entre les nœuds de la plate-forme. Les stations de travail peuvent également, participer à la formation des grilles de calcul. Si l'ensemble des nœuds de la grille est formé par ces ordinateurs de bureau, nous avons affaire au calcul de grille sur stations de travail (desktop grid computing). Les grilles de stations de travail fonctionnent en général, selon le paradigme maître/esclave. Un serveur central possède les codes et les données et il a pour tâche de les répartir entre les esclaves.

- Les grilles de données : une grille de données est une collection de ressources distribuées, mise en œuvre pour traiter et transférer d'énormes quantités de données. Les grilles de fouilles de données entrent par exemple dans cette catégorie.

- Les grilles de services : une grille de services rassemble une collection de ressources distribuées qui procurent des services ne pouvant pas être fournis par un seul ordinateur. C'est par exemple le cas d'un courtier de ressources stockées qui rend possible le partage et la gestion de données et de métadonnées, dans un environnement hétérogène. Ce service est obtenu par l'intégration de différentes bases de données qui appartiennent à différentes entités virtuelles. Les grilles de services sont généralement accédées à l'aide de portails internet (GEONgrid, CHRONOS, NEESGrid, BIRN ou MyGrid). Ces portails cachent les détails et la complexité sous-jacente de la grille.

3.1.2 Caractéristiques des grilles de calcul

Le nom de grille a été emprunté, par analogie, aux réseaux de distribution d'électricité. L'idée est de fournir la puissance de calcul et de traitement de données aussi facilement qu'on peut le faire avec ceux-ci. Pour atteindre ce but, les infrastructures sous-jacentes aux grilles doivent autoriser un accès transparent aux ressources. Elles doivent se montrer pervasives (ubiquitaires) et posséder différents niveaux de sécurité et de fiabilité pour l'accès aux ressources et le traitement des erreurs. Il existe en réalité, plusieurs types de grilles : public ou privé, local ou global, spécifique à des domaines particuliers ou généraliste. Ces grilles ont des objectifs réalistes qui ne prétendent pas traiter tous les problèmes.

Les grilles sont maintenant passées du domaine de la recherche à celui de l'exploitation intensive. Il y a encore dix ans, les infrastructures étaient localisées dans des emplacements connus et bien identifiés. Aujourd'hui, grâce à l'évolution de la technologie, la situation a changé de façon radicale. Les applications sont exécutées sur des plates-formes bien plus hétérogènes que dans le cas des grappes de calcul. Leurs configurations exactes ne sont pas connues à l'avance et présentent les caractéristiques suivantes :

- **Des domaines d'administration multiples et autonomes :** les ressources disponibles sont gérées à travers différents domaines informatiques et différentes entreprises. Les propriétaires gèrent ces ressources de manière autonome en respectant des politiques et contraintes qui leur sont propres. Celles-ci peuvent donc apparaître ou disparaître sans préavis, durant l'exécution des applications.

- **Une extensibilité du nombre de nœuds** : Cette propriété découle directement du premier point. Pendant les traitements, une grille peut passer de quelques nœuds disponibles, à des milliers, voire des millions, et inversement. Cette extensibilité pose des problèmes de performances. Les applications désireuses d'exploiter un grand

nombre de nœuds distribués géographiquement, devront se montrer peu gourmandes en termes de latence du réseau et de bande passante.

- **Une tolérance aux pannes :** les défaillances de nœuds dans les grilles doivent plutôt être érigées en règle qu'en exception. Le nombre important de ressources fait, en effet, augmenter leurs probabilités d'apparition. Pour exploiter efficacement les ressources disponibles, les applications doivent présenter un caractère auto-adaptatif, tel que nous l'avions évoqué dans le précédent chapitre.

La vision de la grille que proposent I. Foster et C. Kesselman en 2004, implique l'utilisation et la mise en œuvre de protocoles, d'interfaces et de politiques, de types ouvert (open source) et standard. Seule, l'utilisation de standards permet de partager les ressources de la grille dynamiquement, et d'éviter une balkanisation de la grille en de multiples systèmes distribués incompatibles entre eux. L'utilisation de standards ouverts peut être considérée aussi, comme une caractéristique essentielle de la grille.

3.1.3 Différences entre calculs sur grappe et sur grille

Bien que les grilles soient perçues comme les héritières naturelles des grappes de calcul, la multiplicité des formes qu'elles peuvent prendre, rend leur nature assez différente. Dans les deux cas, des processus coopérants forment une application et sont exécutés sur un système faiblement couplé d'ordinateurs. Dans les deux cas, la structure des applications exécutées dans ces environnements, est très similaire : les processus qui les composent, possèdent des besoins en calculs et en ressources à satisfaire par les nœuds. La question qui se pose, est la suivante : Comment acquérir les ressources nécessaires pour l'exécution de cette application? Sur ce point, apparaît une différence majeure entre les grappes et les grilles. Elle vient du fait que les grappes sont basées sur la possession des ressources alors que les grilles sont basées sur leur partage. Dans le cas de la grappe, les ressources sont gérées de manière centralisée et tous les nœuds coopèrent pour former une machine virtuelle unifiée. Dans le cas des grilles, chaque nœud est un site d'exécution possédant son propre gestionnaire de ressources dont l'objectif n'est pas de favoriser cette vue unique de la plate-forme.

La distinction entre les grappes et les grilles repose sur cette différence de traitements dans les ressources. L'utilisateur de la grille peut accéder à plus de ressources qu'il n'en possède. Par une analyse théorique, Z. Németh [NEM03]. montre que cette différence se retrouve au niveau sémantique, pour le problème du mappage des éléments de l'application sur les ressources physiques. La grappe crée une machine virtuelle dont les composants ne sont qu'une représentation différente de la plate-forme physique. La grille quant à elle, crée une couche virtuelle qui incorpore à la fois les abstractions des utilisateurs et celles des ressources. Par l'abstraction de l'utilisateur, des comptes locaux sont associés à l'utilisateur qui possède les droits nécessaires pour utiliser les ressources dans un groupe virtuel de ressources. L'abstraction des ressources implique une sélection de ressources virtuelles qui sont effectivement mappées sur les ressources physiques réelles. Les différences fondamentales entre les grappes et les grilles sont donc sémantiques. Le tableau suivant dresse l'état des lieux de ces différences.

Grappes de calcul	Grilles
Composées par un ensemble de nœuds de calcul.	Composées par un ensemble de ressources.
L'utilisateur possède les droits d'accès sur l'ensemble des nœuds de la plate-forme.	L'utilisateur possède les droits d'accès sur l'ensemble de la plate-forme, mais pas sur les nœuds individuels.
L'accès à un nœud donne les droits d'accès à toutes les ressources du nœud.	L'accès aux ressources du nœud peut être restreint.
L'utilisateur connaît les capacités et les caractéristiques de la configuration du nœud.	L'utilisateur ne connaît pas les ressources ou a peu d'informations sur celles-ci.
Les nœuds appartiennent à un seul domaine.	Les ressources peuvent s'étendre sur plusieurs domaines.

Tableau : Comparaison entre les grappes de calcul et les grilles

L'analyse du tableau confirme que les grilles se focalisent sur l'utilisateur. C'est donc là, que réside la différence essentielle. Les applications destinées aux grappes de calcul sont développées par et pour le propriétaire de la plate-forme, pour maximiser l'utilisation des ressources. Dans le calcul sur grille, les machines utilisées pour exécuter une application, sont choisies du point de vue de l'utilisateur, pour maximiser les performances de celles-ci, sans se préoccuper des effets induits sur l'ensemble de la plate-forme. Cette différence rend le calcul sur grille plus abordable pour l'utilisateur, car il n'a pas à se préoccuper de la gestion de la plate-forme.

Il existe également des différences techniques entre les grappes et les grilles d'ordinateurs :

- Dans le cas des grilles, l'utilisateur ne possède pas de connaissances spécifiques à la configuration de la plate-forme et de ses nœuds. Il faut par conséquent, mettre en œuvre un système d'information capable de pallier ce problème. Ce système doit être global et apte à détecter les ressources sur la plate-forme.

- En complément du système d'information global, il faut qu'un système local puisse lui fournir des informations sur l'état et la disponibilité des ressources du nœud concerné.

3.2 Programmation des grilles

3.2.1 Le passage à grande échelle des modèles MPI, RPC et RMI

La notion de passage à l'échelle des applications sur grappe, concerne leurs adaptations aux grilles de calcul. Le portage des applications basées sur RPC et MPI constitue les premières tentatives effectuées dans ce domaine. Avec ces modèles, les processus s'exécutent dans des espaces d'adressage différents et communiquent par échange de messages de manière bidirectionnelle. Chaque processus se comporte à la fois comme un émetteur et comme un récepteur de message. Certaines applications se prêtent mieux que d'autres, à ce type d'exécution à grande échelle sur grille : elles sont distribuées par nature; elles correspondent à

des classes particulières de problèmes tels que le calcul du génome humain ou le calcul et l'affichage d'images et de vidéos. D'autres applications demandent plus d'efforts de conception pour pouvoir fonctionner sur la grille. Une exploitation directe du parallélisme explicite impose une certaine lourdeur dans le développement d'application. Le programmeur doit gérer lui-même les communications et l'équilibrage de charge entre des nœuds de différentes puissances de la grille. Un modèle semi-automatique se révèle mieux adapté dans cette gestion des communications entre processus, qui peuvent se révéler très coûteuses à la longue.

Les modèles de programmation sous-jacents des technologies MPI et RPC ont donc dû être revisités. En conséquence, de nouvelles interfaces de programmation (API) standardisées et de nouvelles librairies pour le lancement des applications, ont été ainsi introduites. Les problèmes d'authentification, d'autorisation et de réservation de ressources à travers les différentes plates-formes et différents domaines d'administration, sont maintenant pris en compte.

Le standard principal pour le paradigme du passage de message dans les grilles, est MPICH. Ce dernier a été développé conjointement par le laboratoire américain d'Argonne et l'Université du Mississippi (USA). MPICH implémente complètement le standard MPI et l'étend avec des fonctionnalités d'entrées/sorties qui ont été définies dans la norme MPI-2 [MPI11]. Les bibliothèques classiques de RMI incorporent des fonctions de communication point à point. Une opération « envoyer » (send) initie le transfert de données entre deux composants d'un programme qui s'exécutent en concurrence. Dans le deuxième composant, une opération recevoir (receive) est exécutée. Elle permet de récupérer les données de la structure de stockage du premier composant. MPI-2 autorise quant à lui, des opérations de type unilatéral. C'est-à-dire qu'une opération « envoyer » n'a pas besoin d'une opération « recevoir » explicite. Ce schéma de communication autorise donc des communications de type asynchrones. Il faut néanmoins que le receveur implémente des mécanismes de scrutation perpétuelle pour détecter l'arrivée de nouveaux messages. La norme MPI-2 supporte donc la gestion dynamique des processus, mais l'auto-adaptabilité doit être encore gérée par le programmeur. MPICH a été conçu comme une architecture d'interface en couches. Au sommet de la pile, figurent les interfaces qui concernent directement les fonctionnalités de MPI et MPI-2. La couche MPICH implémente ensuite ces interfaces, pour fournir d'autres services comme la gestion des erreurs et la manipulation de différents objets cachés. Une couche supplémentaire, nommée ADI (Abstract Device Interface) s'occupe de la migration de données entre les couches MPI et les systèmes réseaux. Cette couche rend l'implémentation de MPI, indépendante du réseau et des systèmes de gestion de processus.

Le standard pour le paradigme de l'appel à distance, RPC (Remote Procédure Call), a été lui aussi amélioré pour pouvoir exploiter la technologie des grilles de calculs. Dérivé directement de ce dernier, le nouveau standard pour les grilles, Grid-RPC (ou Grid-based RPC) [GRI11] constitue ainsi, un autre candidat pour la programmation de celles-ci. En plus des services de base d'appel à distance, il fournit des services de plus haut niveau. Ceux-ci permettent de traiter des tâches asynchrones à moyen et gros grains de parallélisme sur des grilles d'ordinateurs pouvant aller de quelques centaines de nœuds à plusieurs milliers. Un système Grid-RPC est basé sur un modèle client/serveur et met à disposition tout un ensemble d'éléments : un composant client, un exécutable à distance, un composant serveur et un service d'information. Les programmes clients peuvent appeler des bibliothèques situées sur des nœuds distants, en utilisant des interfaces (APIs) spécifiquement fournies. Tous les mécanismes nécessaires à la réalisation d'appels à distance sont inclus dans les bibliothèques

et les interfaces. Le service d'information fournit les éléments nécessaires pour invoquer un exécutable à distance et communiquer avec lui. S'appuyant sur ces mécanismes de base, Grid-RPC cache aux applications, la nature dynamique, instable, et non sécurisée des plates-formes. Il gère dynamiquement la découverte de nouvelles ressources, la tolérance aux pannes et la sécurité (authentification sur les sites, et délégation d'authentification, adaptation aux politiques de sécurité). S'il permet également d'équilibrer les charges et de gérer la malléabilité, Grid-RPC s'en remet cependant complètement aux programmeurs, pour la gestion de la migration des composants de l'application. Il possède encore d'autres caractéristiques telles que la gestion de type de données par langage (IDL : Interface Description Langage). Cette spécificité lui permet de passer en paramètre des matrices entières ou par morceaux, et des fichiers.

A notre connaissance, aucune démarche similaire à celles de Grid-RPC et MPICH, n'a été proposée pour le système d'invocation de méthodes à distance en Java (RMI). Néanmoins, le couple Java-RMI structure les applications en un ensemble d'objets distribués pouvant communiquer à l'aide d'invocations à distance de méthodes. Il propose de cette façon dès l'origine, une démarche de conception très puissante qui sépare clairement l'appelant de l'appelé. Les interfaces de programmations que proposent RMI peuvent donc facilement passer l'échelle, à condition de leur adjoindre des fonctions supplémentaires pour gérer une grille. RMI tout comme ses deux concurrents, peut ainsi, fonctionner au-dessus d'autres intergiciels comme Legion ou Condor, et d'autres encore que nous présenterons ultérieurement. Ils récupérèrent ainsi, des fonctionnalités qu'ils ne peuvent pas offrir eux-mêmes.

3.2.2 Gestion de la grille par l'application

Les applications développées à l'aide de MPICH et GridRPC sont le plus souvent mises en œuvre à l'aide des paradigmes de programmation tels que ceux que nous avons décrits dans le premier chapitre. Traditionnellement, les composants de l'application communiquent à l'aide de protocoles et de mécanismes d'interaction ad hoc. Ils sont répartis dans une architecture comportant cinq couches (voir figure 2) :

- Fabrique : Cette couche représente l'architecture physique de la grille. Elle contient toutes les ressources partagées qui n'incluent pas seulement les ressources de calcul (stations de travail, supercalculateurs), mais aussi leurs systèmes d'exploitation, les ressources de stockage de données, les bases de données et des outils scientifiques comme des capteurs ou des télescopes.

- Connectivité : Cette couche définit les protocoles de communication et de sécurité qui gèrent les échanges de données entre les différentes ressources. Le protocole de sécurité fournit les mécanismes nécessaires pour la sécurisation des échanges et l'identification des utilisateurs et des ressources. Réduisant la complexité du système, les protocoles de cette couche sont basés sur des standards existants.

- Ressource : Cette couche utilise les services communication et sécurité et fournit deux services sécurisés sous forme d'APIs :

- le service d'information réunit des informations sur l'état des ressources partagées,

- le service de gestion alloue et gère l'accès et les droits aux ressources partagées. Il assure aussi une qualité de service, et le monitoring de ces dernières permet de garder un certain contrôle.

- Collective : Cette couche rassemble la plupart des protocoles et des services qui collectent des informations sur les interactions existant entre les ressources. Un grand nombre de services se retrouve dans cette couche : co-allocation et ordonnancement des ressources, monitoring, analyse et diagnostiques (détection des erreurs, des cas de surcharge de calcul), exploration de ressources matérielles (Directory Service) et logicielles, réplication de données.

- Application : cette dernière couche contient les applications utilisateurs. Pour gérer et découvrir les ressources ou accéder aux données, les applications doivent faire appel aux services des couches précédentes par l'intermédiaire d'interfaces de programmation (API) bien définies.

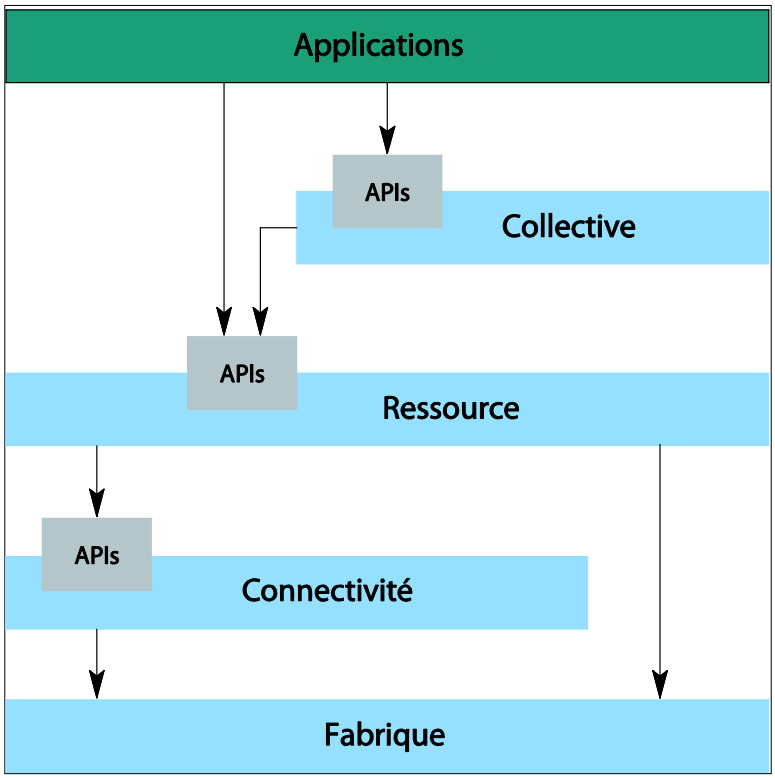

Figure 2 : Architecture d'une application dédiée aux grilles de calcul

Pour pouvoir fonctionner sur une grille, les applications directement développées au-dessus de cette architecture, doivent donc être conçues en intégrant dès le départ, les services et les protocoles proposés par elle. Les développeurs sont donc directement responsables de la

gestion du parallélisme et de la localisation sur la grille des différents composants. Cette approche ne permet donc pas de bénéficier d'une réelle virtualisation des ressources.

3.2.3 Gestion de la grille par des intergiciels

Comme pour les grappes d'ordinateurs, le recours à des intergiciels ou à des frameworks permet d'exploiter pleinement la puissance des grilles, tout en évitant une conception difficile des applications selon les étapes données par le schéma du paragraphe précédent. Les intergiciels peuvent directement être implémentés au sommet de la pile dressée, dans le cadre de cette architecture en couche. Ils présentent des interfaces similaires sous forme d'API, mais avec des fonctionnalités atteignant des niveaux d'abstraction plus élevés. Les services procurés sont des métaservices tels que le courtage de ressources, le monitoring sur l'ensemble de la plate-forme, ou une sécurité accrue. A travers ces services mis à disposition, le programmeur dispose d'une couche virtuelle qui lui permet de s'abstraire complètement de la complexité de la grille.

Globus [GLOB11] constitue un excellent exemple pour ce type d'intergiciels. Il est issu d'un projet américain développé par plusieurs institutions et fournit les services de base nécessaires pour gérer la grille. Ces services sont délivrés aux clients grâce à des échanges de messages. En encapsulant les opérations dans ces échanges, le service isole l'utilisateur des détails de location du service et de son implémentation. Le gestionnaire d'allocation des ressources de la grille (GRAM : Grid Resource Allocation Manager) est le composant de la boîte à outils (toolkit) de Globus qui est en charge de cette gestion des ressources. Il permet de lancer à distance des travaux à exécuter. L'utilisateur met en œuvre la partie « fabrique » qui est présente dans le schéma de conception des applications pour les grilles (cf. paragraphe 3.2.2), par l'intermédiaire des interfaces de GRAM. Cependant l'implémentation actuelle de GRAM rend difficile le déploiement de données complexes et d'applications de calcul intensif.

Associées à Globus, les technologies de communication comme MPICH et GridRPC, ne sont plus maintenant de simples outils de programmation, mais des services à part entière de l'intergiciel. MPICH-G2 intègre les fonctionnalités de gestion de Globus et de MPICH que nous avons évoquées dans le paragraphe précédent. Il propose maintenant des services comme l'authentification, l'autorisation, la création et le contrôle de processus, le monitoring, la redirection des communications, des entrées/sorties et l'accès à distance de fichiers. L'asynchronisme hérité de MPICH peut être mis à profit pour gérer la latence dans les grands réseaux formés par les grilles. MPICH-G2 gère ceux-ci de manière hiérarchique. Il n'est pas la seule implémentation de MPICH destinée à des grilles hétérogènes. Cependant, c'est le seul à cacher et à gérer autant cette hétérogénéité.

Le passage à l'échelle des applications se fait plus aisément en utilisant l'ensemble des services proposés par des intergiciels comme MPICH-G2, grâce aux appels à leurs interfaces de programmation. Le nombre disponible des services génériques et « prêts-à-l'emploi », ne cesse de croître. Ces derniers possèdent en commun l'idée de fournir des fonctionnalités variées, qui capturent des schémas communs de composition ayant recours aux paradigmes de programmation. L'utilisation de ces services permet aux programmeurs de bénéficier de toute la puissance des grilles, tout en étant affranchis de l'écriture de codes spécifiques qui ont été conçus de manière indépendante de l'application. Par exemple, le kit Java COG (Java Commodity Grid Toolkit) [JCG11] fournit un framework orienté «objet» qui intègre les interfaces et les services de Globus. En utilisant ce type de framework, la conception

d'applications pour les grilles, s'avère bien plus aisée. Ce dernier virtualise les ressources de la grille, à travers la mise en œuvre de services qui sont offerts au moyen d'API particulières. L'inconvénient est qu'il requiert la connaissance d'un modèle de programmation et de ces API nécessaires au développement d'applications. Répondant à ce problème, une autre catégorie d'approches fournit des outils de développement ne nécessitant pas de connaissances préalables. Ces approches permettent aux programmeurs de se focaliser uniquement sur les fonctionnalités propres à leurs domaines et à leur métier. Elles permettent de faire passer l'échelle à des applications qui n'avaient pas été initialement prévues pour cela. Elles traitent les applications comme des boîtes noires en se servant, soit de leurs sources, soit de leurs codes machines exécutables. Englober ces derniers dans des composants permet de cacher les détails de la plate-forme. Ces approches amènent néanmoins à un parallélisme à gros grains. De plus, elles conduisent à exploiter des applications monolithiques qui ne peuvent pas tirer tous les avantages d'une grille, contrairement aux approches basées sur les API.

3.3 Conception d'applications distribuées sur grille

3.3.1 Le développement à base de composants logiciels

Les analyses que nous avons réalisées précédemment, montrent un aspect plus prononcé de l'hétérogénéité et de la dynamicité dans la composition des grilles, par rapport à celui des grappes de calcul. À condition de savoir gérer à moindre coût ces deux paramètres, l'utilisateur peut disposer de formidables ressources de calcul. Le problème est de pouvoir développer le plus facilement possible, des applications pour ce type d'environnement. Les modèles de programmation que nous avons présentés dans les paragraphes précédents, permettent effectivement de développer des programmes très efficaces pour les grilles. Cependant, le développement d'applications plus lourdes nécessite l'emploi d'outils logiciels adéquats qui soient capables de gérer à la fois la complexité du support d'exécution et celle du code de l'application développée. Les modèles basés sur les composants logiciels répondent parfaitement à cette exigence. Ils peuvent garantir un parfait niveau de programmation par la composition des applications, l'interopérabilité et la réutilisation des logiciels. Ils sont capables en outre, de fournir des résultats de performances d'exécution de très haut niveau, et de s'adapter à l'évolution de toutes les technologies sous-jacentes (réseaux, nœuds de calculs, systèmes d'exploitation et intergiciels).

Dans ces conditions, les objectifs des systèmes à base de composants, correspondent bien au cahier des charges que nous nous étions fixé pour le développement d'applications à grande échelle pour la plate-forme ADAJ :

- Permettre une complète interopérabilité entre les différents composants de l'application qui auraient été développés à des moments différents.
- Favoriser la réutilisation et l'intégration de codes déjà développés pour d'autres applications et en particulier, du code de notre plate-forme ADAJ elle-même. Les applications peuvent ainsi bénéficier d'une optimisation des performances d'exécution grâce à l'auto-adaptativité que procurent les mécanismes de notre plate-forme.

- Fournir au programmeur des outils pour l'aider dans la conception d'applications et l'exploitation optimale du parallélisme, par la mise en œuvre de primitives spécifiques pour exprimer celui-ci.

Le développement d'une application selon une approche « composants », consiste à la décrire en termes de composants interconnectés au moyen d'interfaces (ports de programmation). Les éléments de base nécessaires pour construire un environnement de programmation de ce type sont le framework et des mécanismes associés de communication et d'interaction entre les composants. Un framework est un kit de composants logiciels structurels qui définissent les fondations ainsi que les grandes lignes de l'organisation de l'application (architecture). Le développement de cette dernière à l'aide du framework, permet de garantir la conformité de tous ses composants à des standards bien établis, ainsi que leurs entières interconnexions. Le framework gère la création, l'instanciation et l'appel des composants à travers toute la plate-forme d'exécution. La plupart des frameworks donnent la possibilité de déclarer un service disponible et de l'invoquer à distance par un autre composant situé sur un autre nœud. Ce mécanisme est basé sur un paradigme client/serveur et repose sur des modèles de type RPC ou RMI tels que nous les avons évoqués précédemment.

Cependant aucun framework existant ne donnait la possibilité de contrôler le comportement parallèle des applications destinées à être exécutées sur une grille. Par conséquent, dans ADAJ, nous avons dû introduire des composants de contrôle du parallélisme des applications et des données, manipulables directement à l'aide du framework. Nous avons par exemple, introduit trois composants de ce type : un composant distributeur, un composant collecteur et un composant pipeline. Le composant distributeur fournit des données aux composants de calcul à chaque connexion alors que le composant collecteur récolte les données, à l'issue des calculs entrepris par ces derniers. Le composant pipeline comporte plusieurs étages et injecte les données en mode pipeline aux composants de calcul. Tous nos composants de contrôle bénéficient de la spécificité de notre framework en termes de dynamicité de création des ports et des composants.

Les applications de calculs scientifiques ont souvent une durée d'exécution très longue pouvant atteindre plusieurs jours, voire plusieurs mois. Pouvoir arrêter ou ajouter dynamiquement des composants pour équilibrer la charge a été le critère privilégié du choix de notre framework. L'environnement de base de celui-ci repose sur les spécifications du modèle CCA (Common Component Architecture). CCA comporte en effet, un système d'événements qui se déclenche lors de la connexion et de la création de composants. Un composant nouvellement créé, peut en temps réel, se connecter aux autres composants de l'application par le biais de la génération automatique de ports. Ce mécanisme nous permet d'offrir un environnement de composants parallèles capables de reconfigurer leurs structures et leurs comportements en cours d'exécution selon des conditions fonctionnelles ou non fonctionnelles (variation dans les performances atteintes). En utilisant le framework de composants, que nous avons développé en y intégrant les outils fournis par la plate-forme ADAJ, le programmeur a la possibilité de contrôler le comportement parallèle de son application sur les grilles de calcul. Nous lui donnons la capacité de se concentrer aussi bien sur le comportement fonctionnel de celle-ci, que sur les aspects qualitatifs de l'exploitation du parallélisme. Mais, il n'a pas la responsabilité de gestion des aspects quantitatifs de cette exploitation qui, eux, sont pris en charge par les mécanismes mis en œuvre dans ADAJ.

3.3.2 Le développement d'applications à base de webservices

Les architectures orientées services (SOA : Service Oriented Architecture) constituent un modèle de programmation permettant de bâtir des applications informatiques flexibles, modulaires et interopérables. Elles constituent une évolution des systèmes basés sur les composants, et les concepts utilisés dans les applications SOA dérivent directement de ces derniers. La différence entre ces deux types de programmation, réside dans le fait que, si jusqu'ici, une application de type composant ne donnait que la description de ses constituants (composants ou objets) et de leurs fonctionnalités, une application basée sur une architecture de type SOA, met au contraire l'accent sur les services rendus.

Un service (webservice) est défini, comme une entité logicielle, en mesure de remplir des tâches données. Les webservices sont basés sur un modèle de programmation de type RPC (Remote Procédure Call). Dans un premier temps, le client cherche parmi les webservices disponibles ceux qui correspondent à ses besoins. Puis, s'il en trouve un, il envoie une requête à ce service qui la traite et retourne une réponse à cette demande. Du point de vue du webservice, la requête est indépendante de toute session et le cycle de vie du service correspond à la durée de la requête. Ils sont sans état. Une fois déployés, les services opèrent indépendamment de l'état des autres services définis à l'intérieur du système. Ils agissent comme des boîtes noires et les composants externes ne savent pas comment ils opèrent pour renvoyer les résultats attendus. Evidemment, ce mode de fonctionnement n'empêche pas les différents services de coopérer sur un objectif commun. L'architecture SOA permet donc d'assembler des applications par morceaux, sans tenir compte des détails de leurs implémentations, de l'endroit où ils sont déployés, et des objectifs initiaux de leurs développements.

De telles applications sont extrêmement utilisées dans les domaines commerciaux, industriels et scientifiques. L'adoption de standards universellement reconnus par les technologies de l'internet (http, XML, SOAP) est le principal attrait des webservices. Mais ce n'est pas le seul cas où ils connaissent ce succès. En effet, les grilles de calcul forment également un cadre idéal pour la mise en œuvre de ce type d'architecture, qui pose les problèmes de déploiement, de gestion et de répartition dynamique des services. Tous les travaux accomplis lors de ces dernières années, montrent que les grilles ne leur fournissent pas seulement, un environnement fructueux, mais amènent un certain nombre de défis à résoudre, et notamment dans le domaine de l'efficacité de l'exécution de ces applications et de l'équilibrage de charge. Dans ces conditions, l'intégration des webservices dans notre plate-forme, prolonge naturellement les travaux que nous avions accomplis jusqu'alors, avec la mise en œuvre du couple Java/RMI.

La standardisation du processus d'intégration demeure l'avantage principal de l'architecture SOA développée pour les grilles. OGSA (Open Grid Services Architecture) [OGS11] est le standard le plus communément accepté. Développé conjointement par plusieurs grandes organisations dont l'OGF (Open Grid Forum) [OGF] et OASIS [OAS11], il apporte aux applications, la souplesse de conception et d'exécution. Dans ce standard, chaque ressource est représentée par un webservice conforme à un ensemble de conventions, notamment en ce qui concerne ses interfaces. L'ajout d'un état au service (WS-Ressource), a permis de faire évoluer OGSA vers un nouveau type de framework : WSRF (Webservice Service- Ressource Framework). L'association des ressources avec état et d'un système de notification des changements d'état de la WS-Ressource, permet de donner naissance à des applications distribuées pérennes et très robustes dans le temps. Nous avons fait évoluer ADAJ pour lui

faire bénéficier de toutes les technologies que nous venons de décrire. Notre nouvelle plate-forme SOAJA (Service Oriented Adaptative Java Applications) permet maintenant, d'instancier des services avec état. La présence d'un état peut être exploitée par les applications clientes et permet les communications asynchrones entre les différents composants de SOAJA, instanciés sur toute la plate-forme.
Beaucoup de types d'architecture et d'intergiciels peuvent être utilisés pour implémenter un environnement basé sur SOA, parmi lesquels on trouve trois systèmes ayant adopté une approche différente :

- SOC (Service-oriented Computing) est un paradigme de calcul qui utilise les services comme éléments fondamentaux pour développer des applications. Les services de base, leurs descriptions ainsi que les opérations de base (publication, découverte/sélection, et la mise en relation entre les services) constituent une couche de base. Celle-ci est complétée par deux couches supplémentaires pour la composition et la gestion de services.

- ESB (Entreprise Service Bus) fournit une infrastructure qui amène une communication robuste, un routage intelligent, une migration et une transformation des services, sophistiquées. Il est basé sur des standards industriels qui facilitent l'interopérabilité. ESB n'est pas un logiciel, mais une façon d'intégrer les applications, de coordonner les ressources et de manipuler les informations.

- SCA (Service Component Architecture) est un ensemble de spécifications libres de droits (open source) qui décrivent un modèle pour développer des applications et des systèmes utilisant SOA. Il permet de cacher lors de l'implémentation et de l'assemblage de services, les détails relatifs à l'infrastructure et aux méthodes d'accès utilisées pour invoquer les services.

Pour notre part, nous avons choisi la technologie middleware ESB (Entreprise Service Bus), qui présente tous les éléments nécessaires pour mener à bien la migration de notre intergiciel et le transformer en une plate-forme performante de type SOA. ESB fournit les supports pour le transport d'information. Il donne aussi facilement accès aux sous-systèmes existants, à travers un ensemble d'adaptateurs. Avec l'aide d'ESB, les services de SOAJA sont exposés de manière uniforme et respectent des standards ouverts, de telle sorte que n'importe quel client peut les utiliser.

3.3.3 Le calcul en nuages

Le calcul en nuages (cloud computing) est un paradigme récent. Il a été popularisé par Amazone en 2006, lorsque cette société américaine a proposé pour un prix de dix dollars par heure d'utilisation, l'accès à des machines virtuelles au moyen d'interfaces utilisateurs et de programmation (API) simples. Ce système (Amazon's Elastic Compute Cloud) fut le premier exemple de nuage public qui offrait un accès payant à l'usage (pay-as-you-use) pour des ressources de calculs disponibles sur internet. La caractéristique principale de ces dernières, repose sur le fait qu'elles sont spécifiquement et dynamiquement créées dans des machines virtuelles, et qu'elles peuvent croître en fonction de la demande. La taille et le nombre de machines virtuelles nécessaires sont évalués dynamiquement par des technologies de virtualisation qui permettent la création, la migration et la destruction de ressources. De cette

façon, le calcul en nuages permet de mettre en œuvre des services sans avoir besoin de connaître l'infrastructure sous-jacente qui les exécute. Pour des raisons marketing, trois types de services peuvent être offerts : le service SaaS (Software as a Service), le service PaaS (Platform as a Service) et le service IaaS (Infrastructure as a Service).

Le modèle SaaS sert à déployer des applications commerciales et donne aux clients, l'impression de posséder leurs propres ordinateurs et logiciels, sans le souci d'achat, de gestion ou de maintenance de ceux-ci. Le SaaS fait donc référence à un modèle de logiciel commercialisé, non pas sous la forme d'un produit (en licence définitive) que le client installerait en interne sur ses serveurs, mais en tant qu'application accessible à distance comme un service, par le biais d'Internet.

Le modèle PaaS fournit des ressources de calcul au travers de plates-formes sur lesquelles les applications et les services peuvent être développés et hébergés. C'est un environnement d'exécution en ligne conçu pour que l'entreprise déploie ses propres applications en dehors de sa salle informatique. Il permet de disposer d'une capacité informatique en constante adéquation avec ses besoins (serveurs, stockage, mémoire...), ou d'applications personnalisables à la demande. Les trois offres les plus importantes sont Force.com de Salesforce, Windows Azure de Microsoft et App Engine de Google. L'inconvénient majeur de l'utilisation du PaaS est au niveau de la portabilité des services, car il n'existe pas encore de standard régissant ces modèles.

La dernière approche (IaaS) aborde le calcul en nuage, par le hardware, afin d'établir un socle souple et robuste pour répondre aux demandes des applications. Elle fournit une infrastructure « à la demande » en fonction des pics de charge dus à l'exécution des applications ou des demandes des utilisateurs. Avec cette infrastructure mutualisée, il devient simple de mettre en place un environnement applicatif, sans passer par de nouveaux serveurs et de nouvelles installations de logiciels. Ce dernier paradigme est le plus intéressant pour nous, car il peut être associé aux grappes et aux grilles de calcul. En général, le nuage IaaS est bâti à partir de trois composants principaux : une couche de virtualisation qui masque l'infrastructure physique (réseaux, ordinateurs et systèmes de stockage), le gestionnaire de l'infrastructure virtuelle, et une interface permettant à l'utilisateur de gérer les machines virtuelles avec de simples abstractions. Il existe aujourd'hui une myriade d'outils et de technologies, qui implémentent l'un ou l'autre de ces composants. Ainsi, des technologies VIM (Virtual Infrastructure Manager) couvrent la partie gestion des machines virtuelles (Platform VM Orchestrator, VMWare DRS ou Ovirt). Un certain nombre d'hyperviseurs leur est associé (KVM, Xen, VMWare). D'autres outils permettent de transformer une infrastructure existante en nuage IaaS comportant des interfaces adaptées (Globus Nimbus, Eucalyptus ou OpenNebula). Comme n'importe quelle infrastructure, les grilles peuvent donc bénéficier de l'avantage apporté par les nuages. L'utilisation de machines virtuelles permet de fournir à l'utilisateur un environnement personnalisé. De plus, la grille peut supporter plusieurs organisations virtuelles avec différentes configurations. Il faut remarquer que l'emploi de machines virtuelles Java (JVM) nous procurait déjà cet avantage.

Pour développer un accès de type nuage dans SOAJA, nous expérimentons l'outil Eucalyptus qui exploite Xen pour la gestion de la virtualisation sur la plate-forme. La particularité d'Eucalyptus est de posséder trois niveaux hiérarchiques de contrôle : le niveau le plus élevé coordonne le nuage tout-entier ; le niveau intermédiaire s'occupe des grappes d'ordinateurs ; et enfin le niveau le plus bas gère les machines virtuelles sur les nœuds physiques de calcul. Avec nos collègues de l'Académie des Sciences de Pologne, nous avons

lancé de nouvelles recherches pour étudier les interactions possibles entre le système d'information de SOAJA et ces différents contrôleurs. Les autres problèmes sont de fournir une qualité de services acceptable et d'introduire de la propriété d'élasticité présente dans les nuages. Cette dernière possibilité trouve une réponse par une gestion dynamique du nombre de JVM disponibles pour effectuer les calculs. Les stratégies mises en œuvre dans ADAJ, pour équilibrer la charge permettent déjà de détecter les machines sous-chargées et les machines surchargées. Il nous suffit par conséquent, d'adjoindre au système d'information, la possibilité de créer ou de découvrir de nouvelles JVM. Ce que nous pensons réaliser avec l'apport des agents mobiles. Ces agents de courtage, sont capables de négocier les ressources et les contrats de qualité de services de type SLA (Service Level Agreement), associés. Ces aspects seront développés dans un prochain paragraphe.

3.4 Les architectures d'extraction de la connaissance (SOKU)

3.4.1 La fouille de données sur grille

Les facilités actuelles de stockage des données dans des bases réparties sur les grilles, rendent très complexes l'analyse et la découverte des informations utiles (connaissances), contenues dans celles-ci. L'extraction de connaissances (Knowledge Dicovery In Databases) est réalisée par un processus particulier qui est la fouille de données (data mining). Il repose sur la découverte de motifs ou de modèles présents dans les données. Celles-ci sont généralement représentées sous forme d'enregistrements pouvant comporter plusieurs centaines, voire des milliers d'attributs. A l'aide de processus particuliers, la fouille de données tente de réduire le volume des données en jouant sur le nombre d'instances ou d'attributs de ces enregistrements. Ces processus sont entre autres :

- la classification qui affecte une classe, à chaque instance d'enregistrement,
- le regroupement d'instances (clustering),
- la recherche de règles d'association qui identifie les implications entre attributs,
- la recherche de séquences ou de similitudes.

Pour chacune de ces tâches, différentes approches issues de différents domaines (statistiques, apprentissage, optimisation combinatoire, …) ont été proposées. La découverte de règles d'association est l'une des techniques phares, que nous avons implémentée dans l'environnement ADAJ. L'analyse du panier de la ménagère est l'un des exemples phares de cette recherche de règles dans le domaine du marketing. Celle-ci porte sur le repérage de produits de consommation que la ménagère pourrait être susceptible d'acheter en complément des achats qu'elle a déjà effectués. La quantité de données à prendre en compte dans ce type de recherche, est si importante, qu'il faut faire appel aux grilles de calcul pour disposer des capacités suffisantes de traitement.

Les grilles amènent cependant, des changements substantiels dans la manière de traiter ces données. Les bases de données peuvent avoir des tailles gigantesques, de l'ordre de plusieurs téraoctets, être implantées géographiquement sur plusieurs domaines administratifs, et les données peuvent être complètement hétérogènes. Les méthodes classiques de fouilles de données deviennent insuffisantes dès lors qu'il s'agit d'explorer ces quantités massives

d'informations. Les problèmes clefs rencontrés se situent au niveau des phases de préparation des données pour l'extraction de règles de connaissances, au niveau de l'extraction proprement dite, mais aussi au niveau de l'interprétation et donc du post-traitement des collections de règles découvertes. Les extractions en particulier, et notamment celles concernant les ensembles fréquents, continuent de soulever beaucoup de problèmes difficiles. En effet, la complexité du calcul est exponentielle quant au nombre d'attributs, et la plupart des applications concernent des milliers d'attributs. Seul un recours à des méthodes d'échantillonnage, permet aux techniques traditionnelles statistiques, d'être utilisables sur d'aussi grandes quantités de données.

L'exigence d'un temps de réponse convenable, nous a conduits à rechercher de nouvelles approches pour le datamining haute performance. Celles-ci reposent sur des heuristiques et non plus sur des méthodes exactes systématiques. Elles permettent à la fois de résoudre de nouveaux problèmes (supports particuliers de règles d'association, recherche multicritères…) et facilitent une exécution parallèle et distribuée sur grilles de calcul. De plus, l'acquisition régulière de quantités d'informations importantes exige de pouvoir travailler de manière incrémentale, et d'étudier la mise en place des structures de données ad hoc associées aux heuristiques. La création d'algorithmes distribués pour du datamining haute performance est donc bien un problème d'actualité, qui nous a donné l'occasion d'étudier la façon d'intégrer le traitement des données dans notre plate-forme orientée service, SOAJA. Les nouvelles heuristiques que nous avons développées ont ainsi pu bénéficier des fonctionnalités développées dans cette plate-forme. Nous avons ainsi montré que les algorithmes de découverte de connaissances incluses dans les données, et les technologies de fouille de données, pouvaient bénéficier largement, de la puissance de calcul apportée par les grilles.

3.4.2 Les services basés sur les connaissances

Le foisonnement des outils de gestion de la grille rend sa maîtrise plutôt complexe. Les développeurs ne perçoivent pas toujours les tenants et les aboutissants de ces différents intergiciels désignés sous des termes et acronymes variés. En particulier, ils peuvent ne pas comprendre à quoi servent ces outils et quelles en sont les propriétés pertinentes. Pour simplifier le développement et l'interopérabilité des applications, il est impérieux de clarifier cette situation. Un groupe d'expert (NGG : Next Generation Grid) travaillant sous l'égide de la commission européenne, a proposé de faire évoluer les grilles de calcul vers un concept plus large de prestations de services, basé sur les connaissances (SOKU : Service Oriented Knowledge Utilities). Les principes généraux de l'architecture mise en œuvre dans les grilles, sont conservés. Celle-ci reste toujours basée sur les webservices et le modèle SOA. Cependant le modèle en couche (pile de logiciels), traditionnellement utilisé dans les grilles, est remplacé par un maillage multidimensionnel, qui conserve les mêmes mécanismes dans chaque dimension. L'accent est mis sur la dynamicité des services et la notion de ressources utiles. Ces dernières concernent des services directement et immédiatement utilisables, avec des fonctionnalités, des performances et une fiabilité bien définies.

La grille n'est pas seulement une infrastructure totalement axée sur les aspects systèmes. Elle doit également rendre possible et faciliter le partage d'information et de connaissance à des niveaux sémantiques plus élevés. Pour supporter l'intégration et la dissémination de ces connaissances, nous avons mis en place la technologie de l'internet sémantique. Ce concept a été introduit pour améliorer la communication entre les différentes technologies, pour

accroître l'interopérabilité entre les bases de données. Il fournit de nouveaux mécanismes qui facilitent la mise en œuvre de services de type SOKU, dans SOAJA. Ces services s'appuient sur les connaissances (information sémantique), pour faciliter le bon fonctionnement de notre plate-forme. En particulier, ils mettent en œuvre des représentations abstraites de la connaissance, les ontologies. Ces dernières modélisent les connaissances en termes de concepts, de relations entre ceux-ci, de hiérarchies et de propriétés de classes. Elles permettent de raisonner sur les représentations de connaissances et offrent un moyen de définir un ensemble d'instances possibles, de concepts et de relations faisant le lien entre le modèle et la réalité. Les ontologies permettent donc de définir une sémantique précise et de partager des terminologies en ôtant toute ambigüité. En particulier, elles sont très utiles, dans le cas de la mise en relation entre les demandeurs et les fournisseurs de service. Deux services peuvent par exemple, avoir les mêmes interfaces d'entrées/sorties, et offrir des traitements diamétralement différents. A l'opposé, deux services ne possédant pas du tout les mêmes interfaces, peuvent réaliser les mêmes traitements. Nous avons résolu, dans SOAJA ces deux types de problème, au moyen de l'approche sémantique et des ontologies. Dans ce cadre, nous avons défini une ontologie particulière à l'aide d'un vocabulaire précis. Ce vocabulaire permet de décrire l'infrastructure de la grille et de spécifier en détail, toutes ses ressources.

Après la mise en relation des services, l'autre grand problème à résoudre par les systèmes de type SOKU, est la découverte des connaissances dans les grilles, déjà évoquée dans le paragraphe précédent sur la fouille de données. Ce problème peut être résolu par la mise en place d'une infrastructure telle que SOAJA, à condition de la compléter, par des services concernant toutes les phases de découverte de la connaissance. Ces services doivent comprendre la gestion des données, la fouille des données, la représentation et la gestion de la connaissance. Pour exploiter véritablement les connaissances, il faut aussi leur adjoindre**,** un ensemble d'ontologies. Ce travail, que nous avons réalisé, nous a permis d'unifier toutes les ressources présentes dans la grille et d'intégrer dans SOAJA la découverte de services. Nous l'avons effectué à l'aide du framework WSRF (Web Service Resource Framework). Cet intergiciel offre un ensemble de standards pour développer des services de type SOKU. L'intérêt de WSRF, est qu'il permet de définir des webservices avec état. Cette évolution majeure des intergiciels facilite le développement des applications pour la grille en mettant en œuvre des services particuliers (WS-Resources). Ce concept de WS-Ressource permet à la fois, de représenter, de faire la publicité, et d'avoir accès aux informations concernant, et le support d'exécution, et l'application elle-même. Pour compléter ce dernier, un mécanisme de publication/souscription a été implanté au travers de la spécification WS-Notification pour tenir informés les clients et/ou services des changements d'état de la WS-Ressource.

L'ensemble des fonctionnalités de WSRF, nous a aidés à bâtir les services de SOAJA, relatifs aux tâches de fouille de données. Ces services adressent toutes les activités depuis la sélection et le transport des données, jusqu'à l'analyse des données, la représentation du modèle de connaissance et la visualisation.

3.4.3 Utilisation des agents mobiles

Les systèmes classiques de fouille de données fonctionnent selon le principe d'une architecture d'application de type 3-tiers constituée par le client, le serveur de données et le serveur de calcul. Chacun de ces derniers possède un rôle particulier, clair et bien défini. Le

client s'occupe de la fouille proprement dite, de la visualisation et des modèles de données. Le serveur gère les données, en contrôle l'accès et coordonne les tâches. Enfin, le serveur de calcul exécute les services de fouilles de données. Cette architecture est simple, mais n'est pas assez dynamique, ni suffisamment autonome pour pouvoir prétendre passer l'échelle de la grille. La solution que nous avons choisie pour pallier ce problème est l'abandon de ce modèle d'architecture de type client/serveur, pour une architecture basée sur les agents. Dans ce modèle, nous avons choisi d'affecter à chaque site de données, plusieurs agents qui analysent les données locales et correspondent avec les agents des autres sites. Ainsi, la connaissance locale élaborée pendant l'opération de fouille de données, peut être partagée et échangée entre les sites pour construire une connaissance globale et cohérente.

D'autre part, comme nous l'avons souligné précédemment, les architectures orientées services permettent de développer des infrastructures innovantes. Ces infrastructures nous ont apporté des services flexibles, tels que la découverte automatique de services ou de nœuds de calcul, la composition intelligente et la migration dynamique de services ou de codes. Y intégrer des systèmes à agents apporte quelques avantages supplémentaires pour la conception des applications ADAJ et l'optimisation de leur exécution. Par définition, un agent mobile est un programme capable de migrer au travers de la grille, en compagnie de son propre code et de son état d'exécution. Les agents sont également en mesure d'exécuter des calculs, là où des ressources ont été allouées. En évitant des interactions coûteuses à distance entre les nœuds, le temps d'exécution est, de ce fait, optimisé. La mobilité permet un mappage dynamique des agents sur l'architecture physique et contribue aussi à améliorer l'équilibrage de la charge entre les nœuds. Enfin, autonomie et proactivité (gestion des effets, plutôt que des causes) peuvent être exploitées pour concevoir de nouvelles stratégies de coopération entre services. Ainsi, associer la mobilité et la composition dynamique de services améliore la disponibilité de ces derniers et peut leur fournir de la valeur ajoutée.

Les agents mobiles nous ont apporté un certain degré de liberté dans la conception des services aussi bien au niveau applicatif, qu'au niveau système. Ils ont facilité l'implémentation de notre intergiciel pour les grilles ADAJ, en fournissant des services efficaces de découverte et de composition, et l'ont complété par des services d'authentification et d'autorisation d'accès des utilisateurs. Cependant, pour supporter complètement le calcul de grille, les agents doivent être en mesure d'assumer différents rôles, être organisés selon des groupes dynamiques régionaux et nationaux, et être capable de migrer entre ces groupes pour supporter l'équilibrage de charge. Gérer l'accès aux ressources de calcul et aux données est une opération complexe et consommatrice de temps. Un système mature implique de pouvoir décider correctement quels systèmes utiliser, où placer les données pour un domaine particulier d'application, comment migrer les données à partir des nœuds de calcul, ou vers eux; ou encore, de savoir quel est le débit de transfert de données exigé pour garantir la bonne exécution de l'application. Nous avons donc retenu pour le développement du système ADAJ, une approche par système de courtage qui donne une réponse assez efficace à toutes ces questions. Basée sur des techniques intelligentes qui mettent en œuvre des agents mobiles, elle nous semble adéquate dans les cas de découverte de services, de gestion de performance, et de sélection de données. Nous avons impliqué les agents mobiles dans trois scénarios importants :

- **La découverte et le courtage des ressources :** les agents mobiles interagissent avec des services de registres distribués pour déterminer l'emplacement des ressources nécessaires (calcul, données, …). Cette approche constitue une alternative avantageuse aux registres UDDI habituellement utilisés par les webservices, ou même à celle mise

en œuvre dans Globus (MDS4). MDS4 comporte deux services de type WSRF : un service index qui collecte les informations sur les données et fournit une interface d'interrogation et de souscription à celles-ci, et un service trigger qui déclenche des actions en fonction de ces données. L'ontologie que nous avons définie, favorise la négociation et le courtage entre les agents et facilite l'accès aux ressources de calcul et aux données.

- **Le monitoring des performances :** les agents mobiles interagissent avec le système d'information d'ADAJ, pour traiter les informations d'observation et distribuer le calcul, au travers des nœuds de la grille. Les stratégies habituelles de détection de la charge et de la puissance des nœuds de calcul sont mises en place par notre système. Mais, en plus de sa stratégie de migration déclenchée par les états des nœuds et des calculs, d'autres stratégies basées sur des modèles prédictifs ont été testés. D'autre part, dans le cadre du calcul vert, le paramètre de consommation électrique du nœud de calcul peut être rajouté. Mais dans ce cas, les algorithmes de choix des nœuds d'ADAJ, doivent être revus pour y intégrer des possibilités de calcul multicritères.

- **La migration de code :** les agents mobiles englobent le code des objets et des composants de l'application ADAJ, pour les faire migrer sur des nœuds distants.

3.5 Etat de l'art des systèmes SOKU et des agents

3.5.1 Fouille de données dans les systèmes SOKU

Quelques systèmes actuels exercent leurs activités de fouille de données, dans des environnements distribués géographiquement. Pour cacher la complexité de la plate-forme et des données hétérogènes, ils ont adopté le standard de grilles OGSA. Les détails de gestion de bas niveaux de la grille sont laissés au soin d'intergiciels, comme Globus. Les architectures de ces systèmes deviennent de plus en plus sophistiquées afin de répondre à la fois, au cahier des charges de la grille, et à l'exigence de fournir à l'utilisateur un accès simple aux ressources de calcul de fouilles de données. A travers WSRF [WSR11], il est possible de définir les services de base de toutes les tâches de fouille de données distribuées, nécessaires à de telles architectures. Ces services concernent tous les aspects, allant de la sélection et du transport des données à leur visualisation, en passant par l'analyse et la modélisation de celles-ci. D. Talia [CON07] dresse une liste exhaustive de services qu'un système est supposé posséder, pour pouvoir prétendre être un framework ouvert pour la fouille de données sur grille. Ces services doivent en particulier comporter des fonctions de prétraitements, de filtrage, de visualisation et de fouilles de données proprement dites (classification, clustering et découverte de règles). Mais ils doivent également inclure des schémas distribués de fouilles de données et être capables de bâtir des applications à partir de graphes de flots complexes. Les données sont manipulées à la volée, grâce à un gestionnaire de graphe fourni par une extension de OGSA, nommée OGSI-DAI (Open Grid Services Architecture - Data Access and Integration). OGSI-DAI [OGS11] est basée sur les webservices et donne un accès uniforme aux ressources de données hétérogènes de bases relationnelles ou xml.

Weka4WS [**CON08**] est un intergiciel qui étend une boîte à outils nommée Weka, à l'aide du framework WSRF. Weka est constituée par un ensemble d'algorithmes d'apprentissage concernant la fouille de données, en Java. Weka4WS gère la distribution des calculs, en exécutant à distance les algorithmes de fouille de données. Pour permettre cette invocation à distance, chaque algorithme de Weka est exposé comme un webservice. L'interface utilisateur de Weka4WS donne le choix entre une exécution locale ou distante des différentes tâches de fouille de données. Weka4WS est programmé en Java, de façon multi-threadée et permet donc de lancer plusieurs tâches à la fois. Sur chaque nœud de calcul, un web service respectant les spécifications de WSRF, donne accès aux algorithmes de fouille de données fournis par les bibliothèques de Weka. Les algorithmes proposés n'étant pas distribués, les processus correspondants sont exécutés sur un seul nœud. Malgré cela, les calculs peuvent quand même, se faire de manière distribuée sur les nœuds de la grille, en exploitant une répartition optimale des données. Dans Weka4WS, les phases de pré-traitement et de visualisation des données sont exécutées localement, alors que les algorithmes de classification, de clustering et de recherche de règles d'association, peuvent être exécutés sur des nœuds distants.

Knowledge Grid (K-Grid) [CON07] est un intergiciel orienté services, offrant un large éventail d'outils relatifs à la fouille de données. Outre ce choix, il donne la possibilité de gérer et de représenter les données sous forme d'abstraction de haut niveau. Dans ce but, le système est organisé en deux niveaux et s'appuie sur Globus pour la gestion de la plate-forme. Le premier niveau est le cœur de K-Grid. Il intègre les services de base pour supporter toutes les phases de la découverte de connaissances : publication, recherche des sources de données et fouilles de données; publication et recherche des outils susceptibles de réaliser ces opérations; définition et gestion de graphes d'exécution de processus complexes de fouille de données, et enfin, présentation des résultats. Chaque service de K-Grid est représenté par un webservice qui exporte une ou plusieurs de ces opérations, selon les mécanismes et les conventions de WSRF. Le second niveau fournit des abstractions et des services supplémentaires, par lesquels il est possible de mieux intégrer les ressources de la grille. Il gère des métadonnées décrivant ces dernières et assure le mappage entre les requêtes abstraites du graphe et les ressources physiques effectivement disponibles. Il veille aussi à la bonne exécution des graphes structurés de fouille de données. L'utilisateur peut ainsi se concentrer sur le développement d'applications complexes de fouille de données, tout en faisant abstraction des détails de bas niveaux de l'infrastructure mise à sa disposition.

L'architecture **Data Mining Grid Architecture (DMGA)** [**PRE07**] utilise trois phases pour la fouille de données : une phase de pré-traitement, une phase de fouille de données (classification, régression, clustering et règles d'association), et une phase de post-traitement. Elle se présente sous forme d'un ensemble de services spécifiques et de schémas génériques d'utilisation et de composition de ces derniers. WekaG est le nom donné à l'implantation effective de cette architecture. Comme la plate-forme Weka4WS, décrite précédemment, elle met en œuvre la boîte à outils Weka, en y ajoutant toutefois, un algorithme de recherche d'instances de type Apriori. Elle est basée sur les services de Globus, pour gérer la plate-forme de calcul et sur ceux de OGSA, pour assurer le développement des services. Le couple DMGA/WekaG permet de combiner et d'assembler différents services fonctionnels de fouille de données (composition horizontale). Il permet aussi la composition de plusieurs instances d'un même service, agissant sur différents ensembles de données (composition verticale). La mise en place d'un protocole de négociation, lui assure les meilleurs choix possibles de services. Finalement, WekaG supporte donc l'implémentation d'algorithmes parallèles de fouilles de données, mais est restreint aux applications développées spécifiquement pour elle.

Le système **DataMiningGrid** [**STA07**] permet d'exécuter sur grille, des applications de fouille de données déjà existantes. Développé à l'aide des librairies de Globus et de WSRF, il permet de distribuer leurs exécutions dans un environnement qui est basé sur l'allocation et la gestion de ressources, effectuées par son propre système de courtage. Il donne la possibilité aux utilisateurs non spécialistes de la grille, de reconstruire facilement leurs applications grâce à son éditeur de flots de tâches. Celui-ci offre un environnement de développement convivial et flexible pour la composition d'application. En particulier, il peut reconfigurer dynamiquement son interface utilisateur, en fonction de certains paramètres donnés par ce dernier. Il autorise également la navigation sur des fichiers distants et peut les transférer entre les différents serveurs de la grille. Il vérifie également les types de chaque composant du flot et de chaque paramètre entré manuellement par l'utilisateur. DataMiningGrid ne gère cependant pas les applications, qui ne peuvent pas être facilement divisées en sous-tâches indépendantes. Il a besoin d'un intergiciel permettant de gérer les communications entre les processus, comme nous pouvons le faire. Les concepteurs de DataMiningGrid ont collaboré avec nous, pour développer notre système SOAJA.

Les systèmes SOKU que nous venons de passer en revue, abordent tous la découverte de la connaissance, par le biais de la construction de graphes de flots, plus ou moins complexes. Certains permettent aussi de gérer interactivement des flots dynamiques, en fonction des besoins de l'utilisateur. De nouvelles spécifications ont même été proposées pour un langage de composition dynamique des services (Dynamic Service Composition Language : DSCL). Ce langage permet de décrire des graphes de tâches composés de services et de spécifier les paramètres correspondants. Un moteur d'exécution (Dynamic Service Control Engine) implémenté sous forme de service, permet ensuite, d'exécuter ces graphes. Ce dernier peut être manipulé interactivement par un utilisateur, pour lancer, arrêter, ou reprendre l'exécution de ces graphes. L'utilisateur peut également changer de graphes ou de paramètres. Ce langage, comme les autres systèmes que nous avons rencontrés, n'apporte cependant pas vraiment de dynamicité dans l'allocation des ressources de calcul et de données, en fonction des variations de la plate-forme d'exécution. La véritable amélioration que nous avons pu apporter, concerne le monitoring, la collecte et la gestion dynamique des informations sur les ressources de la grille. Les applications voient leurs performances augmenter, avec cette possibilité d'équilibrer la charge dynamiquement, sur toute la plate-forme.

3.5.2 Les agents dans les systèmes SOKU

Jusqu'à présent, les systèmes distribués de fouilles de données procédaient en deux étapes pour effectuer leurs tâches. Ils employaient d'abord un ou plusieurs agents pour analyser et modéliser les ensembles de données locales. Ce qui a abouti à faire générer des modèles locaux, par des agents individuels. Ces modèles peuvent être ensuite combinés et synthétisés en un ou plusieurs modèles globaux basés sur différents algorithmes d'apprentissage. Nous pouvons donner en exemple, plusieurs systèmes qui sont développés en Java, suivant ce principe :

JAM (Java Agent for Metalearning) [**PRO00**] est un système de fouille distribuée, développé autour de techniques de méta-apprentissage. Il est utilisé pour détecter les comportements délictueux, dans le domaine financier. Il est architecturé autour de plusieurs bases appartenant à différents instituts bancaires et sur lesquelles coopèrent des agents d'apprentissage et des agents travaillant au niveau méta. Les agents travaillant au niveau des bases de données locales, génèrent des classificateurs locaux. Ces derniers sont dans un

second temps, regroupés sur une plate-forme de données, où par des méthodes de méta-apprentissage, ils peuvent être agrégés en un modèle global.

De manière similaire, le système **BODHI (Beseizing knOwledge through Distributed Heterogeneous Induction)** [**KAR1999**] a été conçu avec l'aide des agents, pour réaliser la fouille de données. Il s'intéresse notamment aux arbres de décision appliqués dans le domaine des spectres de Fourier. Il manipule des processus de fouille de données distribuées entre des sites qui stockent des données hétérogènes. Il modélise ces données par une représentation indépendante. Le système a été bâti sur le principe que chaque processus est composé de manière distribuée, à partir de fonctions de base. BODHI permet aux processus de communiquer facilement entre eux. Pour cela, il gère des agents comme les objets de base et peut les transférer d'un site à l'autre. Ces migrations incluent tout l'environnement de l'agent, son état courant, et les données de l'apprentissage, donc les connaissances acquises.

L'intergiciel **DMGCE (Datamining Grid Computing Environment)** [**LUO07**] s'intéresse au problème de l'ordonnancement des tâches de fouille de données, sur une grille. Sa solution repose sur la mise en œuvre d'un système multi-agents. Il modélise le graphe de flots de tâches par un DAG (Directed Acyclic Graph), sur lequel il réalise un ordonnancement, en deux phases : un ordonnancement à l'échelle de la grille et un ordonnancement local. Ce découpage se retrouve au niveau du graphe de flots et réduit sa complexité : une partie concerne le niveau local compris dans un seul domaine d'administration de la plate-forme (interne à la grille), alors que l'autre s'occupe d'un territoire multi domaine (externe à la grille) réparti sur tout l'internet. Dans la partie interne (IntraGrid), les éléments de calcul et de stockage sont supposés être reliés par un réseau rapide, avec une bande passante garantie. La partie externe rassemble quant à elle, plusieurs IntraGrid. Dans la partie IntraGrid, DMGCE implémente des algorithmes de classification, avec toutes leurs phases de prétraitement (normalisation, discrétisation et réduction du nombre d'attributs). DMGCE a été développé comme un système multi-agents à l'aide de l'environnement MAGE (Multi-AGent Environment) [ZHO03]. Des algorithmes de fouille de données préexistants sont englobés dans les agents. L'ordonnancement interne revient à la mise en compétition de ces agents, qui représentent de ce fait, des portions à exécuter, du DAG général. L'ordonnancement externe est réalisé à l'aide d'enchères faites par des agents, et qui portent sur des critères comme le coût des communications ou la crédibilité des IntraGrids.

AGrIP (Agent Grid Intelligence Platform) [**LUO07-1**] est une infrastructure de fouille de données sur grille qui elle aussi, est basée sur les agents. L'architecture d'AGrIP est structurée en quatre couches relatives à l'infrastructure matérielle, aux agents, à l'environnement de développement et à l'application elle-même. La couche infrastructure comporte les équipements matériels et les bases de données disponibles et répartis sur internet. L'environnement des agents est basé sur l'environnement multi agents MAGE qui apporte différents outils pour concevoir facilement une application, à l'aide des agents mobiles. Cette couche agent constitue le cœur de la plate-forme AGrIP. Elle s'occupe de la location et de l'allocation des ressources, de l'authentification et de l'accès unifié aux ressources, de la communication, de l'assignation des tâches, et de la gestion des bibliothèques d'agents. La couche de développement fournit les outils nécessaires à la création d'agents, à la récupération des données, à la fouille de données, aux raisonnements, et aux systèmes experts. Finalement, la couche de l'application offre des services qui reconfigurent automatiquement les agents selon les applications envisagées (e-commerce, e-gouvernement, e-éducation, fourniture d'énergie et de pétrole).

Les concepteurs de cette dernière plate-forme AGrIP, ont visiblement étendu les services de Globus et suggéré l'utilisation du framework WSRF, mais ne semblent pas l'avoir implémenté. Ces exemples ne sont pas des systèmes de type orientés services pour l'extraction de la connaissance (SOKU). Mais, ils donnent néanmoins, quelques pistes pour développer de tels systèmes. Ils montrent qu'une architecture de fouille de données doit pouvoir supporter l'extraction de connaissance, à partir de larges bases de données distribuées. Cependant, gérer l'accès aux ressources de calcul et de données est une tâche complexe et dévoreuse de temps. Un système mature devrait pouvoir décider quels systèmes utiliser, où stocker les données pour un domaine d'application donné, comment migrer les données pour améliorer les performances de calcul, et la vitesse de transfert de données exigée pour maintenir un taux de performances significatif. Pour atteindre tous ces objectifs, des approches mettant en œuvre le courtage de ressources, sont nécessaires. Elles doivent être basées sur des techniques intelligentes pour supporter la découverte de services, la gestion des performances et la sélection de données.

L'intégration d'agents logiciels dans les architectures SOA, apporte de nouvelles facilités pour la création de plates-formes basées sur la découverte de connaissance (SOKU). Ces agents améliorent tout à la fois, la gestion, l'accessibilité et la mise à disposition des services. Ils peuvent aussi bien encapsuler des éléments systèmes pour gérer une plate-forme, que la logique applicative pour développer des systèmes à base de découverte de connaissances (KDD) et de fouilles de données. Du point de vue système, les agents fournissent sur certaines infrastructures, les mécanismes de découverte et de composition de services, d'authentification et d'autorisation nécessaires pour les utilisateurs, ou de gestion de session. Toutes ces raisons nous ont amenés à introduire la gestion par agents dans SOAJA, même si nous n'avons pas trouvé d'exemple de plate-forme SOKU qui mette vraiment en œuvre des agents mobiles.

3.5.3 Equilibrage de la charge par les agents

Les agents peuvent donc être employés à tous les niveaux. S'ils ont démontré une certaine efficacité, dans le domaine de la fouille de données, ils peuvent être également, très utiles pour équilibrer la charge de la grille. La nature très hétérogène et dynamique dans le temps, de celle-ci, rend difficile le travail du gestionnaire des ressources. Le recours à des agents mobiles, lui permet de répondre à certains critères comme l'adaptativité, l'extensibilité ou le passage à l'échelle. Les exemples de systèmes démontrant cette possibilité, ne manquent pas.

ARMS [CAO02] est un gestionnaire de ressources de la grille, qui est basé sur les agents. Il utilise les agents pour la publication et la découverte des ressources et met en œuvre des mécanismes de prédiction de performances qui étaient fournis par la boîte à outils PACE (Performance Analysis and Characterise Environnement). Une hiérarchie d'agents homogènes et coopérants, a pour but de dispatcher et d'ordonnancer l'exécution des applications sur les ressources disponibles. Chaque agent représente une ressource de la grille et PACE maintient une liste des caractéristiques de tout le matériel disponible. Celle-ci est fournie à chaque agent. Ce qui lui permet de bâtir sa propre table de capacités (Agent Capability Table), où chaque ligne contient une information sur l'identité d'un agent, et des informations sur les services et les performances de la ressource correspondante. Les tables sont mises à jour régulièrement, aussi bien par des méthodes push (l'agent reçoit l'information), que pull (l'agent va chercher l'information). Un modèle de performances et

d'exigences pour son exécution (temps de début, de terminaison, …) est associé à chaque application devant être exécutée. Les agents communiquent, collaborent et négocient entre eux pour réaliser les meilleures performances possibles pour son exécution. Ce projet est aujourd'hui arrêté.

Bond [BOL00] est un intergiciel orienté objet, en Java. Il a été développé pour donner naissance à une infrastructure destinée à un laboratoire virtuel. Il permet d'ordonnancer des tâches complexes et d'annoter des données pour le compte d'applications de calculs intensifs portant sur les données. Il favorise le travail collaboratif en gérant la connaissance, par des graphes de flots. Ses fonctionnalités sont basées sur un système d'objets distribués qui lui permettent de stocker et de manipuler l'information. Les objets s'échangent les informations sur les ressources, par messages et à l'aide d'un langage propre à cette plate-forme. La dissémination de cette information est réalisée périodiquement par une méthode push. Un mécanisme permet à chaque agent de prendre connaissance de la présence des autres. En particulier, chacun d'eux maintient une liste de nœuds actifs, avec lesquels il a déjà eu à faire, et la propage aux autres agents. C'est de cette manière que l'information sur les agents présents dans le système est effectuée. Pour finir, l'ordonnancement de tâches est décentralisé et utilise des modèles prédictifs de coûts. Il faut noter que malgré l'intérêt de cette approche basée sur l'intergiciel PACE, Bond ne connait plus de nouveaux développements.

ASF (Agent-based Scheduling Framework) [SAK07] est une autre approche basée sur les agents, pour ordonnancer les tâches sur les grilles de calcul. ASF est composée d'un méta-ordonnanceur et d'agents autonomes qui sont attachés aux ressources de calcul. L'idée principale est de réduire les responsabilités des ordonnanceurs conventionnels, pour éviter que leur charge de travail ne s'accroisse avec le nombre de ressources disponibles. Pour atteindre cet objectif, ASF utilise des agents autonomes qui sont chargés de rechercher les tâches à effectuer. Elle peut ainsi collecter en continu, des informations sur l'état des ressources de la plate-forme. Elle peut aussi choisir sa politique d'ordonnancement et affecter les tâches, là où elle le veut. ASF a été implémentée à l'aide de l'intergiciel Globus. Des expériences ont montré un gain de 11% de temps d'exécution des applications, sur cette plate-forme.

MAGDA (Mobile Agent-based Grid Architecture) [**AVE06**] est une boîte à outils en Java, qui implémente, elle aussi, les agents mobiles. Son but est de dépasser les limitations que connaissent les intergiciels destinés aux grilles de calcul (faible tolérance aux pannes, pas de possibilité de migration des tâches, …). MAGDA permet la découverte et le monitoring de ressources, ainsi que l'équilibrage de la charge. Elle est basée sur une architecture en couche qui a été développée à l'aide des webservices et de la plate-forme open-source JADE [JAD11]. Il est donc possible de lui ajouter des services ou de l'intégrer à d'autres systèmes aux fonctionnements similaires. L'équilibrage de la charge se fait au niveau de l'application, au moyen d'un agent coordinateur. Cet agent gère les listes de travailleurs et de ressources de calcul qui sont disponibles. Quand un agent débute un travail, le coordinateur met à jour ses listes et enregistre des informations sur l'état de calculs et de leur vitesse relative. Ainsi, ce dernier peut détecter des déséquilibres de la charge dans le système, et peut demander à des agents surchargés, de partager leur charge, avec les moins chargés d'entre eux**.**

3.6 Résumé de nos contributions

Pour simplifier le développement d'applications distribuées et parallèles, tout en profitant d'une exécution efficace, nous avons ajouté un framework de composants (références [24],

[31], [32] et [36]) à la plate-forme ADAJ. Basé sur le modèle de composants CCA (Common Component Architecture), ce dernier incorpore tous ses mécanismes : observation, placement et migration d'objets. Au moment du démarrage de la thèse de Iyad Alshabani [ALS06], CCA était le seul framework complet et disponible qui permettait de disposer d'un large éventail d'applications scientifiques. Cependant, comme il présentait quelques lacunes, il a fallu introduire de nouvelles structures, pour développer ce framework : composition hiérarchique, super-composant et composant distant. Le super-composant peut contenir plusieurs composants de base ou super-composants. Le composant distant permet de bénéficier de toute l'efficacité de distribution et de transparence du modèle de communication présent dans ADAJ. L'intégration de ces concepts permet de profiter pleinement des fonctionnalités de ce dernier, sans affecter la compatibilité avec les autres frameworks CCA. Enfin, pour faciliter la construction d'applications, nous avons introduit une série supplémentaire de composants spécifiques : les composants parallèles essentiellement basés sur le super-composant : le distributeur, le collecteur et le pipeline. L'introduction de ces outils par le biais du framework, aide le développeur à structurer son application parallèle plus aisément. Les surcoûts induits à l'exécution sont compensés par les facilités de conception qu'apporte ce framework.

Ces résultats ont ensuite été transférés au niveau de l'intergiciel SOAJA (Service Oriented Adaptative Java Applications). Celui-ci reprend toutes les fonctionnalités du précédent framework, en y ajoutant une couche de webservices. Il gagne ainsi les propriétés spécifiques à l'architecture SOA (Service Oriented Architecture). Il accède aux données et fait partager les ressources disponibles sur la grille, grâce à la technologie ESB (Entreprise Service Bus). La nouvelle couche ESB permet de déployer les services sur la grille. Elle offre les supports nécessaires pour les interconnexions du transport et expose les sous-systèmes existants par un ensemble d'adaptateurs. SOAJA (références [13], [14] et [41]). quant à lui, met en œuvre les services d'équilibrage de charge, à l'aide du standard WSRF. Le bus ESB rend ces services accessibles de manière uniforme et transparente, sur toute l'infrastructure. A travers les services d'orchestration, SOAJA permet l'exécution d'applications très complexes, réparties sur l'ensemble de la grille. Il est également possible, de faire exécuter une application directement par l'environnement DG-ADAJ sous-jacent, sur un cluster donné. Ces deux modes de fonctionnement cachent les détails des différents environnements de la grille, à l'aide d'un ensemble de webservices. SOAJA offre ainsi un support pour l'intégration et l'interopérabilité des applications. Il rend leurs exécutions efficaces, grâce à l'équilibrage dynamique et transparent de la charge qu'il propose. Enfin, il permet d'instancier des services avec état. Ces états peuvent être exploités par les applications pour rendre asynchrones les communications entre leurs différents composants instanciés sur toute la plate-forme. Notre environnement permet ainsi de conforter des applications suivant le paradigme « programming-in-the-large », nécessaire pour assurer le développement et supporter des processus asynchrones à longue durée de vie.

Ces travaux ont été complétés par un système d'information, basé sur les agents permettant la découverte des ressources disponibles sur la plate-forme (références [2] à [5], [35] à [39] et [43]). Ce travail a également été réalisé dans le cadre du PICS (Programme International de Coopération Scientifique) établi entre le CNRS et l'Académie des Sciences de Pologne et ayant pour cadre le "développement de méthodes heuristiques pour l'optimisation de l'exécution des programmes parallèles". Il a permis d'ajouter un système de courtage de ressources de calcul. Celui-ci comporte la description de ces ressources et l'appariement automatique entre les besoins en calcul et les disponibilités dynamiques des nœuds de la plate-forme. Il a été développé à l'aide de la plate-forme open-source JADE. Des agents travaillent en équipe et interagissent à un haut niveau du système d'information de la plate-

forme ADAJ. Ils sont responsables du courtage des ressources en s'occupant du placement des tâches de calcul, et de leurs bonnes exécutions. A des fins d'exploitation industrielle de notre plate-forme, il est possible d'ajouter au système de courtage, un modèle économique permettant de faire payer l'usage des nœuds de calcul. En retour, nous assurons une qualité de service (QoS) sous forme de contrat de type SLA (Service Agreement Level) établi entre le fournisseur de la ressource et son utilisateur potentiel (références [4].et [5]). Ce contrat définit la portée, les conditions et la période de validité de l'accord. Il est implémenté à l'aide du protocole WS-Agreement (Web Service Agreement). C'est le moyen d'offrir flexibilité, scalabilité et utilisation optimum des ressources, et par conséquent d'augmenter la fiabilité de la grille. Enfin, pour favoriser les négociations dans un contexte de grilles très hétérogènes, nous avons construit une ontologie particulière. Dérivée de celle proposée dans le projet européen CoreGRID [COR11], elle permet de décrire toute l'infrastructure de la grille, en spécifiant les ressources disponibles de manière détaillée, à l'aide d'un ensemble de concepts prédéterminés. Les agents de courtage associés à cette ontologie renforcent l'optimisation de l'équilibrage de la charge de la plate-forme SOAJA. A l'équilibrage de charge bas niveau effectué à l'échelle des JVMS et des objets Java, vient s'ajouter maintenant, un équilibrage de haut niveau, obtenu à travers des négociations autonomes pour l'obtention de ressources de calcul.

L'exploitation de standards libres de droits, ouvre les fonctionnalités de SOAJA aux applications utilisatrices. Pour démontrer cette possibilité, nous avons mis en place dans le cadre de la thèse de Valérie Fiolet [FIO06], une application de fouille de données DISDAMIN (Distributed Data Mining). Cette application a été développée suivant le principe des architectures SOKU et a bénéficié des possibilités offertes par la plate-forme SOAJA (références [12] et de [33] à [37]). L'extraction de connaissances dans des bases de données de très grande taille, nous a conduits à étudier de nouvelles approches capables de répondre aux exigences d'obtention rapide des résultats, tout en exploitant le potentiel de calcul apporté par la grille. Pour résoudre le problème de la recherche de règles d'association dans un contexte distribué, DISDAMIN propose une nouvelle approche s'appuyant sur un partitionnement intelligent des données. Le schéma général se base sur une phase de clustering, progressif (Clustering Distribué Progressif). Le résultat de cette fragmentation est ensuite exploité par un nouvel algorithme distribué de génération de règles d'association (nommé DICCoop), permettant de traiter de manière coopérative et asynchrone ces fragments. A l'aide de SOAJA, les différentes étapes proposées par DISDAMIN sont exécutées de façon indépendante de la plate-forme, selon une orchestration bien définie. Le projet DISDAMIN a été transféré dans le domaine de la distribution et du commerce du futur et dans un contexte d'informatique ubiquitaire.

Bilan et perspectives

1 Bilan

Ce manuscrit donne la description des orientations prises et des travaux effectués lors de nos recherches menées depuis une dizaine d'années, au sein de l'équipe PALOMA du LIFL. Une partie de ceux-ci a été réalisée dans le cadre de trois thèses que j'ai eu la chance de co-encadrer. Ces travaux ont été également réalisés au cours de coopérations scientifiques avec des universités étrangères (Pologne, Italie, Roumanie). En particulier, les travaux sur l'ordonnancement des tâches dans les grilles d'ordinateurs ont été faits dans le cadre d'un programme de coopération entre le CNRS et l'Académie des Sciences de Pologne (PICS), dont je suis le coordinateur. De nombreux stages de recherche (DEA, DESS, élèves ingénieurs ou masters) en France et à l'étranger (Pologne et Italie) ont permis de développer les concepts et les plates-formes proposés dans le cadre de ces recherches. Ainsi, grâce à l'aide d'élèves ingénieurs de l'école Polytech'Lille, une plate-forme a été transférée dans le domaine de la grande distribution, au cours d'un projet que j'ai dirigé et qui a été labellisé par le pôle de compétitivité "Industries du commerce" (PICOM).

Le périmètre de nos travaux s'inscrit dans le domaine du calcul distribué. Nous nous sommes en particulier, intéressés à l'exécution d'applications réparties de très grande taille et dont certains éléments peuvent s'exécuter en parallèle. L'adoption dès le départ, des machines virtuelles Java, nous a permis de pouvoir proposer des solutions génériques qui permettent de s'abstraire du type, de la nature et de la composition du support d'exécution. C'était à l'époque le meilleur choix possible de virtualisation. Dix ans après, l'arrivée des plates-formes de calcul en nuages, nous a confortés dans ce choix. Les concepts architecturaux dans le développement d'applications, sur lesquels nous avons travaillé, sont suffisamment généraux pour être utilisés sur n'importe quel support d'exécution à mémoires distribuées. Grâce aux machines virtuelles, les mécanismes que nous avons introduits peuvent s'exécuter sur des cibles aussi diverses que les grappes, les grilles d'ordinateurs ou les systèmes embarqués. La seule condition reste de pouvoir gérer correctement ces plates-formes. Une partie de notre travail a consisté à définir, à concevoir ce système de gestion des informations et à l'intégrer dans un intergiciel, pour qu'il puisse être facilement utilisé par tout utilisateur. A cette fin, nous nous sommes essentiellement appuyés sur les agents mobiles, mais d'autres techniques ont aussi été explorées dans le cadre du PICS. Que ce soit pour les grappes ou pour les grilles de calcul, nous nous sommes toujours appuyés sur la mise en œuvre de logiciels libres de droits, dans la mesure où ceux-ci existaient.

Le choix technologique de Java et des machines virtuelles offre d'autres avantages. Il s'était imposé à nous, grâce à des recherches préalables de l'équipe concernant les systèmes distribués à base d'objets. Il est apparu qu'en Java, les objets associés à la programmation multi-threadée, permettaient d'obtenir un très bon environnement pour un large degré de parallélisme lors de l'exécution des applications. Dérivées de la programmation par objets, les technologies de composants facilitent encore plus l'intégration d'applications de plus grandes tailles, mais ne changent rien, du point de vue de l'exécution. Au final, celle-ci porte toujours sur des threads ou des objets. De la même façon, les webservices apportent des commodités supplémentaires dans la conception des applications, mais ne gardent pas de

spécificités propres pour leurs exécutions. Toutefois, il est important de souligner que la mise en œuvre de ces technologies de composants et de webservices reste indispensable, puisque sans elles la conception d'applications pour les grilles, serait difficile, voire impossible.

Le second fil conducteur de nos recherches réside dans la poursuite de l'efficacité de l'exécution des applications réparties collaboratives. La mise en œuvre des webservices et des agents mobiles, a permis de faire passer l'échelle à la plate-forme ADAJ. En particulier, cette mise en œuvre allie les critères d'interopérabilité de cette technologie de webservices, à ceux d'auto-adaptativité présents dans nos précédents systèmes, qui étaient destinés aux grappes de calcul. Dans les deux cas, grappe ou grille de calcul, ces paramètres d'efficacité et d'auto-adaptativité reposent sur les mécanismes de base qui ont été introduits dès le départ, dans la plate-forme initiale. Les techniques développées, pour observer, mesurer et gérer l'activité des applications portent sur les objets et les processus légers qui s'exécutent dans chaque machine virtuelle Java. Celles-ci s'appuient donc sur des algorithmes et des paramètres techniques constants et indépendants de la plate-forme et des applications. Elles sont donc réutilisables dès lors que l'on dispose de machines virtuelles fonctionnant avec des processus légers, même s'il ne s'agit pas de machines virtuelles Java. Pour obtenir la propriété d'élasticité propre à la technologie de calcul en nuages, une voie intéressante à suivre, serait de mettre en œuvre les concepts que nous avons développés, sur des machines virtuelles appartenant à ce type de système. Une telle association permettrait de créer ou de supprimer des machines virtuelles, à la demande. Cette possibilité de gestion de machines virtuelles, nous semble très intéressante pour les futures générations de systèmes embarqués que nous plaçons dans nos perspectives.

2 Perspectives

Le ralentissement de la loi de Moore et sa fin prévue d'ici une décennie, apportent quelques défis intéressants. Nous sommes actuellement passés d'un doublement du nombre de transistors tous les deux ans, à tous les trois ans. La raison majeure de ce ralentissement et de cette fin programmée, est essentiellement due aux problèmes de dissipation thermique et de surchauffe des composants électroniques. Malgré ce changement, l'augmentation du nombre de circuits sur une puce électronique de type CMOS reste très appréciable, à tel point que déjà, sur certaines puces électroniques, tous les transistors disponibles ne peuvent pas être exploités au même moment. C'est le cas des puces de type « dark silicon » que prévoit de fabriquer la société britannique ARM (anciennement Advanced RISC Machines). Pour préparer la relève des circuits CMOS, plusieurs technologies sont à l'étude actuellement. Elles prévoient une intégration sans précédent du nombre de processeurs dans une puce. Les systèmes embarqués pourront intégrer l'équivalent matériel des grappes et des grilles de calcul. Ce seront des systèmes à ressources de calculs multiples, parallèles, hétérogènes et reconfigurables. Les tendances de recherche sur les puces se tournent de plus en plus vers des systèmes auto-adaptatifs. Par exemple, dans un système multi-cœurs, il est déjà possible de choisir les cœurs les plus adéquats pour exécuter un morceau de code. Ces systèmes auto-adaptatifs mettent en avant des processus d'auto-tuning. Ceux-ci sont basés sur des techniques d'auto-apprentissage et peuvent être réalisés avant la compilation, à l'installation ou à l'exécution du logiciel. Enfin, plus le taux d'intégration de circuits sur une puce est important, plus le taux de circuits défectueux le devient également. Un circuit étiqueté « trois cœurs » est par exemple, un circuit qui comportait quatre cœurs, mais dont l'un d'entre eux reste défectueux. En raison de ces problèmes d'intégration et de dissipation thermique des composants, il est prévu que la fiabilité des puces diminue encore. Les mécanismes de

tolérance aux pannes et de qualité de services que nous avons développés pour les grilles de calcul pourront dans ces conditions être mises en œuvre, à différents niveaux. D'autres mécanismes présents dans ADAJ deviennent nécessaires. Ainsi, la redondance des calculs, la migration de calculs en cas de panne et l'adaptation du comportement en modes dégradés, pourront être réutilisés.

Nous voulons orienter nos recherches vers des problèmes plus spécifiques à l'exploitation efficaces des «systèmes-sur-puce». Nous nous fixons comme objectif, la création de nouveaux algorithmes d'optimisation, avec toujours comme optique, l'amélioration de l'efficacité de l'exécution des programmes distribués sur ces nouveaux supports. Comme pour les grilles, cette amélioration passe par une meilleure utilisation des ressources matérielles disponibles. Les critères d'optimisation peuvent être variés : fonctionnalités propres de chaque circuit, charge, puissance et consommation d'énergie des circuits composant les puces. Les méthodes de communication et de la composition structurelle des systèmes sur puces, doivent également être prises en compte. Face à cette multiplication de critères, la mise en œuvre d'approches méta-heuristiques basées sur des méthodes d'optimisation multicritères, devient indispensable. Nous pourrons toujours, nous appuyer sur les résultats que nous avons obtenus dans le cas des grappes hétérogènes de processeurs et des grilles de calcul. En effet, la réduction du temps d'exécution des logiciels va de pair avec une recherche de l'optimisation de l'utilisation des ressources de la plate-forme. Cette optimisation passe par une meilleure exploitation de la mémoire distribuée : mémoire globale, mémoire des nœuds ou des caches des processeurs contenus sur la puce. Nous pensons donc, que les outils de monitoring, ainsi que les systèmes d'information que nous avons développés dans nos précédentes recherches, seront très utiles pour atteindre ces objectifs. Enfin, les systèmes distribués sur les futures puces amèneront de nouveaux mécanismes architecturaux et aspects de programmation dont il faudra également tenir compte. Les nouvelles heuristiques d'optimisation devront tirer partie des propriétés, présentes à la fois dans les programmes et dans les architectures les supportant.

Toutes ces recherches futures cadrent parfaitement avec les objectifs préconisés par le réseau européen HiPEAC (High Performance and Embedded Architecture and Compilation) et les industriels. Un fabricant comme Intel, annonce par exemple, que le problème d'optimiser les performances des processeurs, n'est plus un problème matériel, mais logiciel. Ces recherches seront faites au sein du nouveau projet '«Emeraude» qui est en cours de montage dans le laboratoire, et qui rassemble des personnes venant de différents horizons (systèmes embarqués, temps réel, systèmes auto adaptatifs).

Bibliographie

Références du chapitre 1

[ANT05] G. Antoniu, L. Boug and M. Jan, An adaptive supportive platform for data sharing on the grid, Scalable Computing: Practice and Experience 6 (3), pp. 45–55, 2005.

[BAR03] M. Bar and MAASK (Maya Anu Asmita Snehal Krushna). Introduction to OpenMosix. 2003, http ://openmosix.sourceforge.net/linux-kongress-2003 openMosix.pdf

[BAS99] J. Basney, M. Livny, Deploying a high throughput computing cluster, in R Buyya (Editor), High Performance Cluster Computing: Architectures and Systems, Prentice Hall PTR, Volume 1, 1999.

[BOU03] A. Bouchi, Proposition d'un mécanisme d'observation dynamique de l'exécution d'applications Java Distribuées, Thèse de Doctorat en Informatique, Lille, Janvier 2003.

[BEO11] site internet : http://www.beowulf.org/

[CHI03] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia, Architecture and performance of an enterprise desktop grid system, Journal of Parallel Distributed Computing, 63(5), pp 597–610, 2003.

[COR11] site internet : http://www.corba.org/

[DEQ03] C. DeQing, T. Chunqiang, S. Brandon, D. Sandhya, and M. L. Scott, Exploiting high-level coherence information to optimize distributed shared state. In Proceedings of the ninth symposium on Principles and Practice of Parallel Programming, pp 131–142. ACM Press, June 2003.

[ERW01] D. W. Erwin, D. F. Snelling, UNICORE: A Grid Computing Environment, Lecture Notes in Computer Science, 2001, Volume 2150, Parallel Processing, pp 825-834, 2001.

[GFS11] Site internet : http://sourceware.org/cluster/gfs/

[GOE04] R. Goeckelmann,.; M. Schoettner; S. Frenz,; P. Schulthess. Plurix, a distributed operating system extending the single system image concept, Electrical and Computer Engineering, Volume: 4, pp 1985-1988 Vol.4, 2004.

[JAV11] Site internet : http://wwwipd.ira.uka.de/JavaParty

[LIA05] T. Y. Liang; C. Y. Wu; J. B. Chang; C. K. Shieh; P. H. Fan, Enabling software DSM system for grid computing, Parallel Architectures, Algorithms and Networks. ISPAN 2005, 8th IEEE International Symposium on Cluster Computing and the Grid , 2005.

[LGC11] Site internet : http://legion.virginia.edu/

[LUS11] Site internet : http://www.lustre.org/

[MOR04] C. Morin.; R. Lottiaux,; G. Vallee.; P. Gallard.; D. Margery; J.-Y Berthou; I.D Scherson; Kerrighed and data parallelism: cluster computing on single system image operating systems, IEEE International Conference on Cluster Computing, pp 277–286, 2004.

[MOR07] C. Morin, XtreemOS: A Grid Operating System Making your Computer Ready for Participating in Virtual Organizations Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, pp 393-402, 2007.

[MPI11] site internet : http://www.mcs.anl.gov/research/projects/mpi/

[NAU68] P. Naur and B. Randell (Editors), Report on a conference sponsored by the. NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

NIE02] R. V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, H. E. Bal, Ibis: an efficient Java-based grid programming environment, JGI '02: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande, 2002.

[NIE10] R. V. Van Nieuwpoort, G. Wrzesińska, Ceriel J. H. Jacobs, H. E. Bal, Satin: A high-level and efficient grid programming model, Transactions on Programming Languages and Systems (TOPLAS), Volume 32 Issue 3, 2010.

[NON11] http://h20338.www2.hp.com/NonStopComputing/cache/590071-0-0-0-121.html

[OMP11] site internet : http://openmp.org/wp/

[OPE11] site internet : http://opengfs.sourceforge.net/

[PRO11] site internet : http://proactive.inria.fr/

[PIN99] E. Pinheiro et R. Bianchini. Nomad : A scalable operating system for clusters of uni- and multiprocessors. In Proceedings of the 1st IEEE International Workshop on Cluster Computing, pp 247-254 1999.

[PVM11] site internet : http://www.csm.ornl.gov/pvm/

[RMI11] site internet : http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[SPR11] site internet : http://www.eecs.berkeley.edu/Research/Projects/CS/sprite/sprite.html

[THA05] D. Thain, T. Tannenbaum and M. Livny, Distributed computing in practice: the Condor experience, Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, pp 323–356, 2005.

[TAN99] A. S. Tanenbaum, R. van Renesse, and H. van Staveren, G. J. Sharp, S. J. Mullender, J. Jansen etand. van Rossum". Experiences with the Amoeba distributed operating system Communications of the ACM, 33(12), pp 46-63, 1990

[YON08] L. Yong, Single System Image with Virtualization Technology for Cluster Computing Environment, IEEE Third International Conference on Convergence and Hybrid Information Technology, 2008.

[WAL04] B. Walker, Open Single System Image (openSSI) Linux Cluster Project, http://openssi.org/ssi-intro.pdf

Références du chapitre 2

[BeBOP] site internet : http://bebop.berkeley.edu

[BOU03] A. Bouchi, Proposition d'un mécanisme d'observation dynamique de l'exécution d'applications Java Distribuées, Thèse de Doctorat en Informatique, Lille, Janvier 2003.

[CHE03] Z. Chen, J. Dongarra, P. Luszczek, and K. Roche, Self Adapting Software for Numerical Linear Algebra and LAPACK for Clusters, Parallel Computing, 2003.

[GLO11] site internet : http://www.globus.org/

[DIE11] site internet : http://graal.ens-lyon.fr/DIET/

[JAD11] site internet : http://jade.tilab.com/

[NEA02] M. O. Neary, P. Cappello, Advanced eager scheduling for Java-based adaptively parallel computing, Proceedings of the joint ACM-ISCOPE conference on Java Grande, 2002.

[OAS11] site internet : http://www.oasis-open.org/

[OGF11] site internet : http://www.gridforum.org/

[OGS11] site internet : http://www.ogsadai.org.uk/

[OSA11] site internet : http://www.globus.org/ogsa/

[PICS11] site internet : http://www.lifl.fr/~olejnik

[RIB01] R. L. Ribler, H. Simitci, D. A. Reed, The Autopilot performance-directed adaptive control system, Future Generation Computer Systems, Elsevier, Volume 18, Issue 1, pp 175-187, 2001.

[SZY04] B. Szymanski, C. Varela, J. Cummings and J. Napolitano, Dynamically Reconfigurable Scientific Computing on Large-Scale Heterogeneous Grids, in Parallel Processing and Applied Mathematics, LNCS, Springer Verlag, Volume 3019, pp 419-430, 2004.

[VAD00] S. Vadhiyar, G. Fagg, and J. Dongarra. Automatically Tuned Collective Communications. In Proceedings of SuperComputing, November 2000.

[VET00] J. S. Vetter, D. A. Reed: Real-time Performance Monitoring, Adaptive Control and Interactive Steering of Computational Grids. The International Journal of High Performance Computing Applications, Vol 14, No. 4, pp. 357-366, 2000.

[WSR11] site internet : http://www.globus.org/wsrf/

Références du chapitre 3

[ALS06] Iyad Alshabani, Composants logiciels distribués et parallèles en Java, Thèse de Doctorat en Informatique, Lille, Décembre 2006.

[AVE06] R. Aversa, B. Di Martino, N. Mazzocca and S. Venticinque**,** MAGDA: A Mobile Agent based Grid Architecture**,** Journal of Grid Computing**,** Volume 4, Issue 4, pp 395-412, 2006.

[BOL00] L. Boloni, K. Jun, K. Palacz, R. Sion, and D. C. Marinescu. 2000. The Bond Agent System and Applications, In Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, D. Kotz and F. Mattern (Eds.), Springer-Verlag, London, UK, 99-112, 2000.

[CAO02] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd. 2002. ARMS: An agent-based resource management system for grid computing. "Scientific. Programming", IOS Press, Amsterdam, Netherland, 10, 2,135-148, 2002.

[CON07] A. Congiusta, D. Talia and P. Trunfio, Distributed data mining services leveraging WSRF, Future Generation Computer Systems , Elsevier, Volume 23, Issue1, pp. 34–41, 2007.

[CON08] A. Congiusta, D. Talia and P. Trunfio, Service-oriented middleware for distributed data mining on the grid, Journal of Parallel and Distributed Computing, Volume 68, Issue 1, Pages 3-15, January 2008.

[FIO06] Valérie Fiolet, Algorithmes distribués d'extraction de connaissance, Thèse de Doctorat en Informatique, Lille, Novembre 2006.

[FOS98] I. Foster and C. Kesselman. eds., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, 1998.

[FOS01] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid, Enabling Scalable Virtual Organizations, International Journal of Supercomputer Applications and High Performance Computing, Volume 11, Issue 3, pages 212-223, 2001.

[GIL00] George F. Gilder, Telecosm: How Infinite Bandwidth Will Revolutionize Our World, The Free Press, NY, 2000.

[GRI11] site internet : https://forge.gridforum.org/projects/gridrpc-wg/

[KAR1999] H. Kargupta, B. Park, D. Hershberger and E. Johnson, Collective data mining: a new perspective toward distributed data mining. In: H. Kargupta and P. Chan, Editors, Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI Press, 1999.

[JAD11] références internet : http://jade.tilab.com/

[JCG11] références internet : http://wiki.cogkit.org/wiki/Main_Page

[LUO07] P. Luo, K. L, Z. Shi and Q. He, Distributed data mining in grid computing environments, Future Generation Computer Systems, Elsevier, Volume 23, Issue 1, pp. 84–91, 2007.

[LUO07-1] Luo, M. Wang, J. Hu and Z. Shi, Distributed data mining on agent grid: issues, platform and development toolkit, Future Generation Computer Systems, Elsevier, Issue 23, Number 1, pp. 61–68, 2007.

[MPI11] site internet : http://www.mpi-forum.org/docs/docs.html

[NEM03] Z. Németh and V. Sunderam, Characterizing Grids: Attributes, Definitions, and Formalisms, Journal of Grid Computing, Klewer Academic Publishers, Volume 1, Number 1, pp 9-23, 2003.

[PRO00] A. Prodomidis, P. Chan, Meta-learning in distributed data mining systems: Issues and approaches, in H. Kargupta, P. Chan (Eds), Book on advances of distributed data mining, MIT/AAAI Press, 2000.

[PRO07] M.S. Prez, A. Snchez, V. Robles, P. Herrero and J.M. Pea, Design and implementation of a data mining grid-aware architecture, Future Generation Computer Systems, Elsevier, Volume 23, Issue 1, pp. 42–47, 2007.

[SAK07] S. Kentaro and S. Hiroyuki, A resource-oriented grid meta-scheduler based on agents. In Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks (PDCN'07). ACTA Press, Anaheim, CA, USA, pp 109-114. 2007.

[STA07] V. Stankovski, M. Swain, V. Kravtsov, T. Niessen, D. Wegener, J. Kindermann and W. Dubitzky, Grid-enabling data mining applications with datamininggrid: an architectural perspective, Future Generation Computer Systems, Elsevier, Volume 24, Issue 4, pp 259-279, 2008.

[ZHO03] S. Zhongzhi; Z. Haijun; D. Mingkai; Z. Zhikung; S. Qiujian; J. Yuncheng; MAGE: multi-agent environment, Computer Networks and Mobile Computing, ICCNMC,. International Conference on Computer Networks and Mobile Computing , pp 181 – 188, 2003

Publications

Livres, chapitres d'ouvrages, éditions d'ouvrages

[1] R. Olejnik, Special section: Grid-like distributed in amorphous networks, Future Generation Computer System (FGCS) - The International Journal of Grid Computing: Theory, methods and Applications, R. Olejnik Ed, Vol 23, issue 8, Springer Verlag, November 2007.

[2] M. Drozdowicz, M. Ganzha, M. Paprzycki, R. Olejnik, I. Lirkov, P. Telegin, M. Senobari, "Ontologies, Agents and the Grid: An Overview", in B.H.V. Topping, P. Iványi, (Editors), "Parallel, Distributed and Grid Computing for Engineering", Saxe-Coburg Publications, Stirlingshire, UK, Chapter 7, pp 117-140, 2009. ISBN 978-1-874672-41-8.

[3] M. Senobari, M. Drozdowicz, M. Ganzha, M. Paprzycki, R. Olejnik, I. Lirkov, P. Telegin, N.M. Charkari, "Resource Management in Grids: Overview and a discussion of a possible approach for an Agent-Based Middleware", in B.H.V. Topping, P. Iványi, (Editors), "Parallel, Distributed and Grid Computing for Engineering", Saxe-Coburg Publications, Stirlingshire, UK, Chapter 8, pp 141-164, 2009, , ISBN 978-1-874672-41-8.

[4] M. Drozdowicz, K. Wasielewska, M. Ganzha, M. Paprzycki, N. Attaoui, I. Lirkov, R. Olejnik, D. Petcu, C. Badica, "Ontology for Contract Negotiations in an Agent-based Grid Resource Management System", in P. Iványi, B.H.V. Topping, (Editors), "Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering", Saxe-Coburg Publications, Stirlingshire, UK, Chapter 15, pp 335-354, 2011, ISSN 1759-3158.

[5] K. Wasielewska, M. Ganzha, M. Paprzycki, M. Drozdowicz, D. Petcu, C. Badica, N. Attaoui, I. Lirkov, R. Olejnik, "Negotiations in an Agent-based Grid Resource Brokering System", in P. Iványi, B.H.V. Topping, (Editors), "Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering", Saxe-Coburg Publications, Stirlingshire, UK, Chapter 16, pp 355-374, 2011, ISSN 1759-3158.

Revues d'audience internationale avec Comité de lecture

[6] A. Bouchi, R. Olejnik and B. Toursel, Java tools for measurement of the machine loads. In Advanced Environments Tool and Applications for Cluster Computing, LNCS 2326, Springer Verlag, pp 271−278, September 2001.

[7] V. Felea, R. Olejnik and B. Toursel. ADAJ: a Java Distributed Environment for Easy Programming Design and Efficient Execution, Schedae Informaticae, Vol 13, pp 9−36, UJ Press, 2004.

[8] E. Laskowski, M. Tudruj, R. Olejnik, B. Toursel, Java programs optimization based on the most-often-used-paths approach, in Parallel Processing and Applied Mathematics, LNCS 3911, pp 944-951, Springer Verlag, Berlin, Heidelberg, 2006.

[9] E. Laskowski, M. Tudruj, R. Olejnik, B. Toursel, Bytecode Scheduling of Java Programs with Branches for Desktop Grid, Future Generation Computer System (FGCS) – The International Journal of Grid Computing: Theory, methods and Applications, R. Olejnik Ed, Vol 23, issue 8, pp. 977-982, Elsevier Science Publishers B. V., November 2007.

[10] R. Olejnik, B. Toursel, M. Ganzha, M. Paprzycki, Combining Software Agents and Grid Middleware, in Advanced in Grid and Pervasive Computing, C. Cerin and K.-C Li Editors, LNCS 4459, pp 678-685, Springer Verlag, Berlin, Heidelberg, 2007.

[11] M. Drozdowicz, M. Ganzha, W. Kuranowski, M. Paprzycki, I. Alshabani, R. Olejnik, M. Taifour, M. Senobari, I. Lirkov, Software Agents in ADAJ: Load Balancing in a Distributed Environment, in: M. Todorov (ed.), Applications of Mathematics in Engineering and Economics'34, American Institute of Physics, College Park, MD, 527-540, December 2008.

[12] V. Fiolet, R. Olejnik, E. Laskowski, Ł. Masko and M. Tudruj, et B. Toursel,. Data Mining on Desktop Grid Platforms, Lecture Notes in Computer Science, Volume 4967, Parallel Processing and Applied Mathematics, Pages 912-921, 2008

[13] R. Olejnik, F. Fortis, B. Toursel, Webservices Oriented Datamining in Knowledge Architecture, Future Generation Computer System (FGCS) – The International Journal of Grid Computing: Theory, methods and Applications, Vol 25, Elsevier Science Publishers B. V., pp 436-444, April 2009.

[14] R. Olejnik, I. Alshabani B. Toursel, E. Laskowski, M. Tudruj, "Load Balancing Metrics for the SOAJA Framework", Scalable Computing: Practice and Experience Journal, pp 419–428, Vol 10, no. 4, December 2009.

[15] M. Ganzha, M. Paprzycki, M. Drozdowicz, M. Senobari, I. Lirkov, S. Ivanovska, R. Olejnik and P. Telegin, Mirroring information within an agent-team-based intelligent Grid middleware; an overview and directions for system development, Scalable Computing: Practice and Experience Journal, pp 397–411, Vol 10, no. 4, December 2009.

[16] M. Ganzha, M. Paprzycki, M. Drozdowicz, M. Senobari, I. Lirkov, S. Ivanovska, R. Olejnik and P. Telegin, Information Flow and Mirroring in an Agent-Based Grid Resource Brokering System, in Large-Scale Scientific Computing, LNCS, Springer Verlag, Heidelberg, pp 475-482, Vol 5910, 2010.

[17] I. De Falco I, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, "Extremal Optimization Approach Applied to Initial Mapping of Distributed Java Programs". Euro-Par 2010, LNCS Vol. 627, pp. 180-191, Springer-Verlag Berlin Heidelberg, 2010.

[18] E. Laskowski, M. Tudruj, I. De Falco I, U. Scafuri, E. Tarantino and R. Olejnik, «Extremal Optimization Applied to Task Scheduling of Distributed Java Programs" , C. Di Chio et al. (Eds.): EvoApplications 2011, Part II, LNCS Vol 6625, pp. 61–70, Springer-Verlag Berlin Heidelberg, 2011.

Colloques internationaux avec Comité de lecture

[19] A. Bouchi, B. Toursel and Olejnik, R. An observation mechanism of distributed objects in Java. 10th EuromicroWorkshop on Parallel Distributed and Network−Based Processing. Las Palmas de Gran Canaria, Spain, IEEE Computer Society, pp 117-122, January 2002.

[20] A. Bouchi, R. Olejnik and B. Toursel. A new estimation method for distributed Java object activity. 16th International Parallel and Distributed Processing Symposium. Marriott Marina, Fort Lauderdale, Florida, IEEE Computer Society, pp 116–121, April 2002.

[21] R. Olejnik, A. Bouchi and B. Toursel. A java object policy for load balancing. PDPTA : The international Conference on Parallel and Distributed Processing Techniques and Applications. Vol. 2, pp 816−821. Las Vegas, USA, June 2002.

[22] R. Olejnik , A. Bouchi and B. Toursel. Object observation for a java adaptative distributed application platform, International Symposium on Parallel Computing in Electrical Engineering (Parelec 06), Poland, IEEE Computer Society, pp 171−176, September 2002.

[23] R. Olejnik, A. Bouchi and B. Toursel, Observation Policy in ADAJ, Accepted to PDCS 2003: Parallel and Distributed Computing and Systems, Los Angeles, USA, October 2003.

[24] Iyad Alshabani, R. Olejnik and B. Toursel, Parallel Tools for a Distributed Component Framework, IEEE International Conference on Information and Communication Technologies: From Theory to Applications, ICCTA'04, Damascus, Syria, IEEE Computer Society, pp 599–600, April 2004.

[25] G. Paroux, B. Toursel, R. Olejnik and V. Felea, A Java CPU calibration tool for load balancing in distributed applications. ISPDC'2004 : The 3rd IEEE ISPDC, Cork, Ireland, IEEE Computer Society, pp 155–159, 5−7 July 2004.

[26] R. Olejnik, E. Laskowski, B. Toursel et M. Tudruj, Java Byte Code Static Scheduling as an Initial Program Optimization Step. Proceeding WP, in 30th EUROMICRO conference, Rennes, France 2004, ISBN 3-902457-05-08.

[27] E. Laskowski, R. Olejnik, B. Toursel et M. Tudruj; Scheduling Byte Code−Defined Data Dependence Graphs of Object Oriented Programs, IEEE International Symposium on Parallel Computing in Electrical Engineering,, PARELEC'2004, Dresden, Germany, 7−10 Sep 2004.

[28] R. Olejnik, E. Laskowski, B. Toursel, M. Tudruj et I. Alshabani, Optimized Java computing as an application for Desktop Grid, "Cracow Grid Workshop '04 Proceedings", Editors: Marian Bubak, Michal Turala, Kazimierz Wiatr, Published by Academic Computer Centre CYFRONET AGH, Cracow, Poland, ISBN 83-915141-4-5, March 2005.

[29] E. Laskowski, M. Tudruj, R. Olejnik, B. Toursel, "Java Byte Code Scheduling Based on the Most-Often-Used-Paths in Programs with Branches, 4th International Symposium on Parallel and Distributed Computing, ISPDC-05, Lille, France, 4-6 July 2005, IEEE Computer Society, ISBN: 0-7695-2434-6.

[30] R. Olejnik, E. Laskowski, B. Toursel, M. Tudruj et I. Alshabani, DG-ADAJ: a Java Computing Platform for Desktop Grid, "Cracow Grid Workshop '05 Proceedings", Editors:

Marian Bubak, Michal Turala, Kazimierz Wiatr, Published by Academic Computer Centre CYFRONET AGH, Cracow, Poland, April 2006.

[31] I. Alshabani, R. Olejnik and B. Toursel. A Framework for Desktop GRID Applications: CCADAJ, IEEE International Conference on Information and Communication Technologies: From Theory to Applications, ICCTA'06, pp 3359-3364, Damascus, Syria, April 2006.

[32] I. Alshabani, R. Olejnik, B. Toursel, E. Laskowski, M. Tudruj, Component-Oriented Programming over GRID with CCADAJ, In Proceedings of International Symposium on Parallel and Distributed Computing (ISPDC 06), Romania, IEEE Computer Society, pp 208-214, July2006.

[33] V. Fiolet, G. Lefait, R. Olejnik, B. Toursel : Optimal Grid Exploitation Algorithms for Data Mining. In Proceedings of International Symposium on Parallel and Distributed Computing (ISPDC 06), Romania, IEEE Computer Society, pp 246-252, July 2006.

[34] V. Fiolet, E. Laskowski, R. Olejnik, L. Masko, B. Toursel, M. Tudruj, Optimizing Distributed Data Mining Applications Based on Object Clustering Methods. International Symposium on Parallel Computing in Electrical Engineering (Parelec 06), IEEE Computer Society, ISBN: 0-7695-2554-7, pp 257-262, Poland, September 2006.

[35] R. Olejnik, B. Toursel, M. Ganzha, M Paprzycki, Utilizing Ressource Brokering Agents in Grid-Bases Data Mining, WP in Euromicro PDP 2007, Napoli, Italia, February 2007.

[36] S. Venticinque, B. Di Martino, R. Aversa, R. Olejnik, B. Toursel, I. Alshabani, Integrating Distributed Component and Mobile Agents programming models in Grid Computing, 6th IEEE International Symposium on Parallel and Distributed Computing, ISPDC07, Hagenberg, Austria, IEEE Computer Society, July 2007.

[37] V. Fiolet, R. Olejnik, B. Toursel, E. Laskowski, M. Tudruj, Data Mining on Desktop Grid Platforms, 7th IEEE International Conference on Parallel Processing and Applied Mathematics PPAM 2007 Gdansk, Poland, September 9-12, 2007.

[38] M. Drozdowicz, M.Ganzha, W. Kuranowski, M.Paprzycki, I. Alshabani, R. Olejnik, M. Taifour, B. Toursel, M. Senobari, I. Lirkov, Software Agents in ADAJ: Load Balancing in a Distributed Environment, 34th Conference on Applications of Mathematics in Engineering and Economics (AMEE '08), Sozopol, Bulgaria, AIP Conference Proceedings, Volume 1067, pp. 527-540, 8-14 June 2008.

[39] M. Senobari, R. Olejnik, M. Drozdowicz, M. Paprzycki, W. Kuranowski, M. Ganzha,, I. Lirkov, Combining an JADE-agent-based Grid infrastructure with the Globus middleware Initial Solution, in: M. Mohammadian (ed.), IEEE International Conference on Computational Intelligence for Modeling Control & Automation (CIMCA IAWITC 2008), IEEE CS Press, Los Alamitos CA, pp : 895-900, 2008.

[40] R. Olejnik, I. Alshabani, B Toursel, E. Laskowski, M. Tudruj, Load balancing in the SOAJA web service platform, IEEE International Multiconference on Computer Science and Information Technology (IMCSIT), Wisla, Poland, IEEE Computer Society, pp 459–465, 20-22 October 2008.

[41] I. Alshabani, R. Olejnik, B. Toursel: Service Oriented Adaptive Java Applications, Middleware 2008, ACM Workshop on Middleware for Service Oriented Computing, MW4SOC 2008, Leuven, Belgium, 1-5 December 2008.

[42] E. Laskowski, I. de Falco, M. Tudruj, R. Olejnik, B. Toursel, "Initial Deployment of Distributed Java Programs in Cluster of JVMs through Extremal Optimization Approach, Cracow Grid Workshop '09 Proceedings", Editors: Marian Bubak, Michal Turala, Kazimierz Wiatr, Published by Academic Computer Centre CYFRONET AGH, Cracow, Poland, pp 44-54, February 2010.

[43] M. Drozdowicz, M. Ganzha, M. Paprzycki, K. Wasielewska, I. Lirkov, R. Olejnik, N. Attaoui, Utilization of Modified CoreGRID Ontology in an Agent-based Grid Resource Management System, Proceedings of the CATA Conference, 2010.

[44] E. Laskowski, M. Tudruj, I. De Falco, U. Scafuri, E. Tarantino, R. Olejnik, B. Toursel, Distributed Java Programs Initial Mapping Based on Extremal Optimization, Para 2010: State of the Art in Scientific and Parallel Computing, Reykjavík, June 6-9, 2010.

[45] I. de Falco, E. Laskowski, R. Olejnik, U Scafuri, E. Tarantino, M. Tudruj, B. Toursel, Extremal Optimization Approach Applied to Initial Mapping of Distributed Java Programs, Europar Conference, Ischia, Naples, Italy, August 31st –September 3rd 2010.