N° d'ordre: 00000

THÈSE

présentée

devant l'Université de Lille 1

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE LILLE 1
Mention INFORMATIQUE

par

Lamine AOUAD

Contribution à l'algorithmique matricielle et évaluation de performances sur les grilles de calcul, vers un modèle de programmation à grande échelle.

Soutenue le 09 décembre 2005 devant la commission d'examen :

MM. :	Jean-Marc	GEIB	Professeur à l'Université de Lille 1	Président
	Michel	DAYDÉ	Professeur à l'ENSEEIHT - Toulouse	Rapporteurs
	Pierre	SENS	Professeur à l'Université de Paris 6	
	Franck	CAPPELLO	Directeur de recherche INRIA	Examinateurs
	Tahar	KECHADI	Professeur à l'Université College Dublin	
	Serge	PETITON	Professeur à l'Université de Lille 1	Directeur de thèse

À la mémoire de mon père.

Merci !

J'ai vraiment beaucoup de chance! Tous les gens que je connais sont adorables et très gentils. Cette thèse n'aurait certainement pas vue le jour sans eux.

Tout d'abord, Merci à toi Serge. Merci pour m'avoir permis de réaliser ce travail, pour ta confiance, ta patience et ta générosité. Ces trois années de thèse étaient très agréables sous ta direction.

Mes sincères remerciements vont aux membres du jury. Merci à Jean-Marc Geib pour avoir accepté de présider mon jury. Merci à Michel Daydé et Pierre Sens pour avoir accepté de rapporter ce travail et d'avoir consacré une partie de leur temps précieux à relire ce manuscrit et pour leurs précieux commentaires et remarques. Merci à Franck Cappello et Tahar Kechadi pour avoir accepté de juger ce travail et pour leurs questions et vision précise du domaine. Un très grand merci à vous pour m'avoir fait l'honneur de votre présence.

Merci aux membres de notre équipe, Pierre, Laurent, Guy et Toussaint pour leur remarquable sérieux. C'était agréable de travailler en votre compagnie. Merci à l'équipe XtremWeb, aux administrateurs de Grid'5000 et à l'équipe de Mme Nahid Emad au PRiSM qui ont contribué d'une façon ou d'une autre à cette thèse.

Enfin, merci à mes parents et mes frères et soeurs pour leur amour et leur soutien de tous les instants.

Table des matières

Chapitre 1

Introduction

Ce chapitre présente brièvement une introduction et la motivation pour la grille. Il résume les éléments clés des grilles de calcul et présente les contributions de cette thèse. En fin de chapitre, l'organisation du reste de ce mémoire est présentée.

1.1 Motivation pour la grille

Récemment, le *Grid computing* ou grille de calcul, a émergé comme un important domaine du calcul parallèle et distribué, distingué du calcul distribué traditionnel par son orientation vers le partage de ressources à grande échelle, dynamiques, géographiquement réparties et hétérogènes. La motivation pour le Grid vient d'un besoin réel en puissance de calcul, notamment pour applications de type "Grand Challenge" ; scientifique, d'ingénierie ou commerciale, qui exigent de plus en plus de ressources (de calcul et de stockage) qu'un seul site ou organisme pourrait en fournir. Des premières applications militaires de déchiffrement de codes ou de simulations nucléaires, jusqu'aux applications récentes en biologie ou en physique des particules, les applications gourmandes en ressources ne manquent pas. La croissance constante de la puissance des ressources et la réserve importante disponible via Internet ainsi que la banalisation du haut débit et des technologies des réseaux rapides donnent l'opportunité de globaliser l'accès aux ressources informatiques largement distribuées dans le monde pour l'exécution d'applications de très grandes tailles. L'ambition du Grid est donc de créer l'illusion d'un grand et puissant "ordinateur virtuel".

L'inspiration du Grid "informatique" vient de la prédominance, la fiabilité et la facilité d'utilisation du réseau de distribution électrique, d'ailleurs le terme Grid trouve son origine dans cette analogie [Fos01]. En utilisant cette association, l'objectif est que la puissance de calcul soit aussi disponible et facile à utiliser que de brancher un appareil à une prise électrique. Depuis 1998, le nombre important de projets commencés autour du Grid témoigne d'une activité de recherche considérable dans ce domaine. Ces projets couvrent souvent plusieurs domaine de recherche : développement d'infrastructures, de middleware et d'environnements de programmation,

outils de simulation et de visualisation, mise en place de bancs d'essai, etc. On peut en citer quelques-uns[1] comme DataTAG, UNICORE, GRACE, et au niveau national, plus d'une vingtaine de projets dans le cadre de l'ACI GRID et l'ACI Masse de données (CGP2P, GRID-ASP, GRID-TLSE, CoDage...).

Parallèlement, il y a eu le développement de plusieurs applications distribuées, dans différents domaines et utilisant des ressources sur Internet. SETI@home [ACK+] (application de radio-astronomie) qui utilise aujourd'hui l'infrastructure BOINC [And] fut l'un des premiers projets dans le domaine du calcul distribué à grande échelle, avec Distributed.net[2]. Ces projets ont réussi à mobiliser des millions d'utilisateurs volontaires et ont permit d'obtenir des puissances de calcul jamais atteintes. On peut citer d'autres exemples de projets orientés application comme Climatprediction.net pour l'étude des changements climatiques, Décrypthon en génomique, ou LHC@home et Einstein@home dans la physique, etc. Par ailleurs, des systèmes intégrés et infrastructures de gestion de ressources sur la grille existent et permettent le déploiement d'applications propriétaires et l'accès aux ressources distribuées, de calcul et de stockage, ou à des instruments d'observation et de visualisation, tels que Globus [Fa03][KTF03], XtremWeb [LFC+03][LC03], NetSolve [SYAD], DIET [CD05][Sut02], Condor [TTL04][FTF+02], et beaucoup d'autres (dont quelques-uns sont décrits dans ce manuscrit).

Le calcul sur la grille pose beaucoup de problèmes et les hypothèses qui étaient plus ou moins valides sur les architectures parallèles traditionnelles ne le sont plus sur la grille. Ainsi, en parallélisme classique on suppose souvent un système "crédible" ; pas ou peu de fautes, des conditions minimales de sécurité, la disponibilité d'informations globales sur le système et une politique simple de partage de ressources. Ces hypothèses ne sont plus valides dès que le système devient largement distribué [Casa].

Dans cette étude, nous qualifierons de distribué un système qui utilise des ressources de calcul hétérogènes qui ne partagent ni la mémoire ni le stockage des données. Un degré supplémentaire de difficulté est alors atteint en ce qui concerne les interconnexions, les communications, l'hétérogénéité des ressources, la sécurité et les performances. Nous remarquons toutefois qu'un système distribué n'est pas forcement parallèle au sens calcul car l'activité n'est pas forcement concurrente et simultanée. Les grilles informatiques tentent de réconcilier le parallèle et le distribué dans le sens où elles visent la haute performance en accédant à des ressources logicielles et matérielles sur des systèmes géographiquement répartis.

La distinction essentielle entre le parallèle et le distribué est que le parallélisme repose généralement sur des restrictions d'architecture, de topologie ou de gestion de

[1] http ://gridcafe.web.cern.ch/gridcafe/
[2] http ://www.distributed.net

la sécurité (souvent sur des machines dédiées et fermées), et vise essentiellement la performance. Tandis que les systèmes distribués sont essentiellement un assemblage hétérogène de ressources. Les approches mises en oeuvre par ces deux systèmes sont différentes. Les grilles de calcul tentent de combiner les aspects des systèmes parallèles et distribués notamment en ce qui concerne les modèles et paradigmes de calcul.

Les expériences menées dans le domaine ces dernières années ont prouvé la faisabilité du partage des ressources à très large échelle. Néanmoins, plusieurs problématiques demeurent dans ce type de calcul. Cela inclut la sécurité, la gestion des ressources (de calcul et de communication), allocation et QoS, l'ordonnancement efficace des tâches, la portabilité, le stockage de masses de données, la fiabilité, l'exactitude des calculs, etc. Aussi ces environnements manquent de services, outils et langages de haut niveau pour le développement d'applications, la recherche de ressources, la gestion d'exécution, ou la négociation d'accès.

1.2 Contexte

Parmi toutes les contraintes du calcul sur la grille, deux sont particulièrement importantes en algèbre linéaire. La première est la performance, i.e. comment utiliser efficacement le grand nombre de ressources distribuées dans la grille afin de réduire les temps d'exécution de ces applications qui sont à priori non-adaptées à ce type de plateformes. Un ordonnancement efficace des tâches est fortement lié à ce premier point. En second lieu, les communications, i.e. la nécessité d'optimiser les communications par un choix judicieux d'emplacement préalable des données et d'anticipations d'envois. Le caractère éventuellement centralisé des communications est aussi un problème majeur dans ce cas. Ces problèmes peuvent être vus et traités à plusieurs niveaux ; au niveau système, au niveau utilisateur ou au niveau application. Au niveau système, des protocoles et services satisfaisant ces dispositions peuvent être développés. Au niveau utilisateur, des APIs (Application Programming Interfaces) de systèmes de grille peuvent proposer les abstractions de programmation requises. Enfin, au niveau application, les caractéristiques de chaque application sont examinées afin de déduire la meilleure approche possible pour une implémentation sur la grille. À terme, les efforts à chaque niveau convergeront vers un calcul sur la grille à haute performance viable et efficace. Dans ce travail de thèse, nous considérons une approche au niveau application. Nous proposerons ainsi un cadre de programmation parallèle et distribuée sur la grille et le testerons sur des applications matricielles à grande échelle.

L'algèbre linéaire et le calcul matriciel ont des propriétés naturellement parallèles. Toutefois, les systèmes de grille actuels implémentent des modèles client-serveur et multi-paramètres mal adaptés à ce type d'applications. L'algèbre matricielle sur la grille nécessite donc d'introduire des optimisations et hypothèses, notamment pour la gestion des communications. Sous ces conditions, nous considé-

rons que les applications matricielles à grande échelle peuvent être déployées efficacement sur la grille. En même temps, ces applications peuvent être utilisées dans le but de traiter les questions importantes du calcul sur ces environnements. Comme le reste des applications sur la grille, les applications matricielles à grande échelle se confrontent aux défis et challenges existants pour une utilisation efficace de tels systèmes. Davantage de recherches sont nécessaires pour traiter, entre autres, les problèmes de performance, d'ordonnancement et de communication et pour appliquer efficacement le calcul en Grid au calcul scientifique et notamment en calcul matriciel. Le but de cette thèse est de tirer profit des caractéristiques des applications matricielles afin de développer une approche de programmation et un cadre de calcul sur la grille pour des implémentations à grande échelle performantes et efficaces.

1.3 Terminologie

Traditionnellement, les systèmes de calcul parallèle sont définis comme un couplage de processeurs pouvant opérer ensemble afin d'acomplir une solution concurrente à une tâche commune. Dans un système parallèle traditionnel, les processeurs sont souvent synchronisés, partagent éventuellement de la mémoire (physiquement ou simulé par des mécanismes de mémoire distribuée partagée) et utilisent un moyen de communication rapide et fiable. Dans ce travail de recherche, nous considérons des systèmes réparties avec couplage lâche. Typiquement, ces systèmes présentent des problèmes différents que les systèmes parallèles traditionnels, car les ressources (de calcul et de communications) sont fortement hétérogènes et autonomes. Techniquement, la signification du *Grid computing* ou *Metacomputing* est proche de cette définition. Ces systèmes peuvent donc être définis comme "des environnements informatiques faiblement couplés partageant des cycles CPU ou des données". Selon leurs fonctionnalitées, les systèmes de Grid peuvent être de calcul (*computational Grid*), d'accès (*access Grid*), ou de stockage et de partage de données (*data Grid* ou *datacentric Grid*) [Ski]. Dans cette thèse, nous nous intéressons aux grilles de calcul et aux applications matricielles à grande échelle.

Rappelons que l'idée clé du calcul sur la grille est d'acquérir et partager du temps de calcul sur les ressources disponibles (universités, industriels ou particuliers) via Internet, ou des liens dédiés, et dispersées à travers le monde pour l'exécution des applications distribuées de très grandes tailles. Actuellement, il n'existe pas de définition précise, communément acceptée, du terme grille ou Grid. Notre contexte est le calcul haute performance sur systèmes distribués à large échelle et notre définition du Grid est la suivante : "l'utilisation coordonnée et extensible des capacités de calcul, communication et stockage pour l'exécution d'applications parallèles ou distribuées sur des ressources hétérogènes situées sur des domaines d'administration (système) différents". Néanmoins, vu la nature "grain fin" des applications traitées, nous visons des grilles assurant une continuité de service avec de bonnes capacités

de communication (si possible), c'est-à-dire des ensembles de ressources (clusters ou en LAN) plutôt que des PCs sur Internet.

1.4 Contribution

Le but de cette recherche est de développer des techniques et fournir des recommandations pour le calcul à grande échelle sur grilles de calcul pour l'algèbre matricielle. Notre approche est d'examiner la nature des applications visées, en termes de volume de calcul et de communications afin de développer des techniques, au niveau application, dans le but d'augmenter les performances. Les objectifs de ce mémoire sont :

1. la recherche et l'analyse des caractéristiques inhérentes à certaines applications matricielles sur les grilles de calcul à travers simulations et expérimentations sur plateformes réelles,
2. développer des approches et techniques au niveau application pour aborder la question de performance et de l'efficacité des calculs,
3. donner un cadre pour le calcul haute performance à grande échelle sur la grille basé sur ces techniques, et l'étendre à d'autres applications,
4. proposer un paradigme de programmation pour la grille basé sur l'exploitation de la localité des données, en utilisant des techniques de placement efficace et de traitement de contraintes mémoire utilisant la technique *out-of-core*,
5. associer un modèle de calcul à ce paradigme de programmation sur la grille pour les applications traitées.

La méthodologie privilégié dans cette étude est l'expérimentation. Deux applications matricielles à grande échelle ont été testées et commentées sur de larges plateformes de calcul déployées sur trois sites géographiquement distribués, en France (Lille et Orsay) et au Japan (Tsukuba), et sur la plateforme nationale Grid'5000[3]. Des analyses théoriques ont été aussi menées à travers simulations et étude de modèles de calcul.

1.5 Organisation du document

Le reste du document est organisé comme suit. Au chapitre 2, nous présentons l'état de l'art du parallélisme, des systèmes de calcul distribués à large échelle et de l'algèbre linéaire parallèle. À partir des paradigmes de calcul, nous présentons les systèmes existants et leurs fonctionnalités. Nous évoquons ensuite les bibliothèques d'algèbre linéaire parallèle. Le chapitre 3 présente la problématique de la gestion des ressources, des données et de l'ordonnancement, qui sont à notre avis, les challenges les plus importants des grilles de calcul.

[3] http ://www.grid5000.org/

Le chapitre 4 présente les outils de simulation pour les grilles et les déploiements et bancs d'essai utilisés dans nos expérimentations. La conception, architecture et administration des outils d'expérimentation sont alors détaillées. Le chapitre 5 présente l'implémentation des applications testés et l'évaluation des performances. Nous montrons que, dans tous les cas, une meilleure exploitation de la localité des données augmente les performances de façon significative. Le chapitre 6 présente le paradigme de programmation pour le calcul matriciel sur la grille basé sur la technique *out-of-core*, ainsi qu'une nouvelle implémentation des applications testés. Nous présentons ensuite une étude des modèles de calcul existants et définissons un modèle associé à nos applications. Le chapitre 7 présente une discussion sur les principaux problèmes et solutions potentielles en calcul scientifique global. Il propose aussi un certains nombre de perspectives à apporter à cette étude. Enfin, le chapitre 8 présente la conclusion du manuscrit.

Chapitre 2

Calcul parallèle et distribué à grande échelle : état de l'art

Dans cet état de l'art, nous présenterons une vue d'ensemble du calcul parallèle et distribué, et des grilles de calcul. Nous commencerons par une introduction au parallélisme et nous présenterons ensuite le calcul sur la grille ou *'Grid computing'* et les approches des différents projets dans le domaine.

2.1 Parallélisme

Le calcul parallèle a pour origine la recherche de la haute performance. Typiquement, transformer un programme séquentiel en un programme parallèle a pour but d'améliorer le temps global d'exécution, ou le passage à l'échelle. Le parallélisme peut être introduit à différents niveaux. Au niveau instruction, par exemple, en utilisant le pipelinage et les composants vectoriels. Néanmoins, le gain de performance est assez restreint par rapport à une exécution purement séquentielle. Il est généralement nécessaire de dupliquer les CPU. Le vrai défi du parallélisme est alors de coordonner efficacement leur travail.

Les systèmes parallèles sont tous formés d'unités de traitement et d'unités de mémoire. La différence tient au nombre et type d'éléments de chaque catégorie et à la façon dont ils sont interconnectés. Historiquement, les premiers systèmes parallèles développés ont été des architectures multiprocesseurs communément appelées "supercalculateurs". Ils sont caractérisées par un grand nombre de processeurs reliés par un réseau interne à haute performance, ou des processeurs exploitant le parallélisme en interne capable de travailler directement sur des vecteurs de nombres (appelées "vectorielles"). Le maître mot de ces architectures est la haute performance. On associe souvent les supercalculateurs, notamment vectorielles, à Cray Research (aujourd'hui division de SGI) et des machines telles que Cray-1 conçu par Seymour Cray. D'autres constructeurs fabriquaient également des supercalculateurs comme

Nec, Fujitsu, IBM ou nCube, etc. Le marché des supercalculateurs est devenu très restreint depuis le début des années 90 et seuls quelques constructeurs comme IBM, Hitachi ou SGI continuent à concevoir ces systèmes. Les autres constructeurs ont été en majorité rachetés et beaucoup ont fait faillite.

Par ailleurs, l'augmentation des puissances de calcul des processeurs grand public, et la disponibilité des matériels réseaux haute performance tels que Gigabit Ethernet, Myrinet ou SCI, ont permis l'émergence des clusters de calcul. Ces architectures composées de simple PC offre indéniablement un meilleur rapport performance/prix que les supercalculateurs. Cette tendance est de plus en plus significative ; plus de 60% des 500 systèmes parallèles les plus puissants au monde sont aujourd'hui des clusters (304 sur 500, alors que leur première apparition ne date que de 1997).

Il y a beaucoup à dire sur les architectures des systèmes parallèles (nature et nombre des éléments de traitement, disposition mémoire, dispositifs d'entrées/sorties, modèles de communication et réseaux d'interconnexion...), nous ne ferons ici que survoler cette problématique. Plusieurs chercheurs se sont essayé à proposer une taxonomie des architectures parallèles. La plus citée, bien qu'elle soit extrêmement rudimentaire, est celle de Flynn. Elle est basée sur les concepts de flux d'instruction et flux de données. Ces deux flux étant indépendants, il y a quatre combinaisons possibles ; SISD (Single Instruction Single Data) correspond au modèle traditionnel des machines séquentielles (machine de Von Neumann). SIMD (Single Instruction Multiple Data), les différents fils d'exécutions exécutent le même flot d'instructions sur des jeux de données différents. Ils peuvent avoir une seule unité de commande mais plusieurs unités arithmétiques et logiques capables d'appliquer l'instruction à plusieurs données. Les architectures vectorielles sont un exemple de ce modèle. MISD (Multiple Instruction Single Data) correspond au modèle connu sous le nom de pipeline. Et enfin, MIMD (Multiple Instruction Multiple Data), des flots d'instructions différents travaillent sur des données différentes, c'est le modèle de la plupart des systèmes parallèles. Cette classification peut être étendue et affinée en considérant le type et disposition mémoire (partagée, distribuée ou hybride). La façon d'accéder à la mémoire pour les architecture à mémoire partagée (uniforme, non uniforme, au travers d'un cache...), etc.

La programmation d'applications parallèles est intimement liée à l'architecture et au type du système parallèle. On distingue généralement deux modèles courants de programmation (suivant le modèle d'exécution SPMD - Single Program Multiple Data) : à mémoire distribuée ou à mémoire partagée. En parallélisme à mémoire distribuée, chaque noeud possède sa propre mémoire. L'accès au données des autres noeuds se fait par des communications (principalement par passage de message ou appel de procédure à distance) en utilisant des primitives logicielles d'envoi et de réception. On se retrouve donc déjà face au problème de répartition des données. En

parallélisme à mémoire partagée, la mémoire physique est commune. Tous les processus partagent un même espace d'adressage mappé sur cette mémoire commune. La mémoire partagée peut être physique ou simulée par des mécanismes logiciels de DSM (Distributed Shared Memory), ou matériels (NUMA). Il s'agit dans ce cas d'architectures hybrides plus extensibles que les architectures à mémoire physiquement partagée.

On parle aussi en général de deux formes de parallélisme ; le parallélisme de données, issue des structures de données manipulées, et le parallélisme de contrôle, issue de la structure de l'application. L'ampleur du potentiel de parallélisme exploitable dépend donc de la taille des structures de données et de ce qu'on appelle la "granularité", définie à partir des algorithmes et en fonction des communications (ou dépendances dans le code). Il existe aussi différentes formes de parallélisation possibles liées aux exécutifs, infrastructures logicielles et compilateurs. Idéalement, le programmeur ne se soucierait pas de la parallélisation du code et laisserait au compilateur le soin de paralléliser automatiquement le code. Il s'agit d'un parallélisme implicite et l'expert ou utilisateur est alors remplacé par des heuristiques et techniques d'analyse de dépendances. Néanmoins, la parallélisation automatique se heurte au problème de l'extraction du parallélisme, et aux limites de la syntaxe et sémantique du langage utilisé. En outre, les langages de programmation parallèles (HPF, OpenMP...) et les librairies parallèles (PVM, MPI...) permettent d'exprimer explicitement le parallélisme du programme avec les mots du langage ou par des appels aux fonction de la librairie. Le développement de logiciels en calcul matriciel est souvent le résultat de parallélisation explicite car il nécessite une connaissance des différents algorithmes répondant aux problèmes étudiés et une approche empirique de la stabilité et complexité.

Il ressort finalement que la priorité des systèmes parallèles est la performance. Les architectures, les modèles de programmation, les standards (tels que MPI ou OpenMP) et infrastructures logiciels vont dans ce sens. Le parallélisme classique repose sur des hypothèse plus restrictives que les systèmes distribués au sens large, et certaines simplifications (d'homogénéité ou de topologie par exemple) sont faites dans le but de permettre certaines optimisations pour une efficacité maximale. La tendance actuelle des grilles de calcul va à l'encontre de ça car prend en compte d'autres facteurs, à savoir l'emergence des réseaux longue distance à haut débit, la disponibilité des machines de plus en plus puissantes et bon marché, des sites de calcul géographiquement répartis, et un coût réduit d'utilisation. Il n'existe pas encore de standard pour ces plateformes et le but ici est l'étude de parallélisation efficace d'algorithmes de calcul matriciel pour ces plateformes quelle que soit la granularité du code.

2.2 *Grid computing*

Le but principal du Grid est de créer l'illusion pour l'utilisateur d'un grand et puissant ordinateur virtuel qui serait la connexion d'une très large collection de systèmes hétérogènes reliés par le réseau. Ces dernières années, le Grid a émergé comme un important domaine du calcul distribué et est motivé par un besoin réel en puissance de calcul. Il y a plusieurs raisons à cela; les stations de travail deviennent de plus en plus puissantes et c'est à même de continuer pendant plusieurs années, ainsi que les technologies et les performances réseau, et l'omniprésence d'Internet. Ces technologies offrent la possibilité d'utiliser des ordinateurs largement distribués pour la résolution de problèmes à grande échelle. Le terme Grid a été choisi par analogie à la grille électrique qui fournit un accès sûr et transparent indépendamment de sa source. On trouve dans la littérature plusieurs appellations : metacomputing, calcul global, grille de calcul, et plus récemment calcul pair-à-pair.

Les grilles permettent donc le partage, l'association et le choix de ressources hétérogènes, dynamiques et généralement non-dédiées, comprenant des clusters, des stations de travail individuelles ou des supercalculateurs (d'ailleurs à la base du Grid on trouve notamment le réseau expérimental I-Way [DFP+96] d'interconnexion de supercalculateurs aux États-Unis). Les ressources peuvent être des systèmes de stockage, des sources de données ou tout autre dispositif de calcul ou de communication géographiquement distribués et appartenant éventuellement à différentes organisations. Par ailleurs, les grilles ne sont pas seulement des infrastructures de calcul pour applications parallèles et distribuées à grande échelle, mais aussi des technologies qui peuvent mettre en relation et unifier diverses ressources allant d'un supercalculateur à un PDA.

2.2.1 Étapes importantes dans l'apparition du Grid

Dans cette section nous parlerons des avancées technologiques et efforts en calcul et interconnexions réseau qui ont mené à l'émergence du Grid. Les plus importantes sont les infrasctructures de communication, et notamment Internet. Le réseau des réseaux trouve son origine en 1969 avec la création d'ARPANET (Advanced Research Projects Agency NETwork), le réseau militaire américain composé au début de quatre noeuds : l'université de Californie à Los Angeles, le Standford Research Institute, l'université de Santa Barbara en Californie et l'université d'Utah. Dans le milieu des années 70, ARPANET est mis à disposition de la communauté, et plus de 30 universités y participent alors, mais reste sous le contrôle des militaires. Le protocole de communication, qui allait devenir le standard sur Internet, TCP/IP, a été proposé en 1978. Dès 1985, le NSF (National Science Foundation) participe avec l'ARPA aux efforts de collaboration entre les centres de supercalculateurs et le monde de la recherche en informatique, et en 1989, la responsabilité et la gestion d'ARPANET ont été officiellement passées au monde académique sous l'égide de la NSF.

L'invention du Web en 1989 au CERN (Centre Européen de Recherche Nucléaire) constitue une révolution majeure dans l'évolution d'Internet. Il a fourni les moyens pour créer et organiser des documents (en utilisant le langage HTML) avec des liens et de leur accéder de manière transparente, indépendamment de leur endroit à l'aide des protocoles, navigateurs et serveurs http. Le W3C (World-Wide Web Consortium) formé en 1994 a été ensuite chargé de développer de nouvelles normes pour l'échange d'information tel que XML (eXtended Markup Language), ou récemment les *Web Services* pour fournir un accès aux applications distantes comme un service.

Les premières applications distribuées ont été exécutées dès le début des années 70 sur le prédécesseur d'Internet, ARPANET. Cela inclut une paire de programmes appelés "*Creeper and Reaper*" qui étaient les premiers programmes sur le modèle de "vol de cycles". C'étaient des vers circulant sur le réseau pour faire des copies ou supprimer d'anciennes copies. En 1973, le Xerox Palo Alto Research Center (PARC) installe le premier réseau Ethernet. Les chercheurs du PARC ont développé un programme appelé *worm* (semblable à "*Creeper and Reaper*") qui circulait sur 100 machines connectées par Ethernet. Ils visualisaient le parcours du *worm* d'une machine à l'autre et récupéraient les ressources inoccupées pour d'autres tâches. Le *worm* circulait dans tout le réseau et avait la capacité de se reproduire et de transmettre des copies à d'autre noeuds dans le réseau. Les premiers efforts de l'informatique répartie étaient donc en cours.

Depuis le milieu des années 1990, la disponibilité d'ordinateurs puissants et des réseaux haut débit, tels que Gigabit Ethernet ou Myrinet, a permit l'émergence des clusters de PCs, qui consituent aujourd'hui les architectures les plus utilisées dans le calcul haute performance selon le classement fourni par le top500[1] (plus de 300). La disponibilité de Linux, un système d'exploitation "open source", puissant et fiable a aussi contribuer à l'émergence des grappes d'ordinateurs (c'est le système préféré de beaucoup de plateformes).

À la fin des années 90, deux tendances du Grid apparaissent ; l'une concentré sur le partage de ressources hautes performance tels que clusters et supercalculateurs, généralement désignée par le terme *Metacomputing*, tandis que l'autre se concentre sur le partage de ressources publiques tels que des PCs personnels reliés à Internet ; pour le calcul (tels que les projets basés sur l'infrastructure BOINC) ou le partage de fichiers (KaZaa, Morphus). Enfin, étant donné le nombre de projets commencés partout dans le monde depuis début 2000, il est clair que la recherche et le déploiement du Grid se développeront significativement dans les années à venir.

[1] http ://www.top500.org/

2.3 Paradigmes de calcul sur la grille

Dans cette section nous présenterons les modèles et méthodes de programmation sur la grille. Contrairement à la programmation parallèle classique, la programmation de la grille doit tenir compte de la nature ouverte, hétérogène et dynamique de l'environnement, que ça soit en terme de ressources impliquées, de leur performance ou de la spécifité de l'application. Il y a actuellement un consensus sur le fait que les modèles et outils existants sont insuffisants pour assurer un développement efficace de codes distribués pour les grilles. Une question importante est aussi posée : quels modèles, abstractions, outils, ou méthodologies de programmation peuvent permettre d'exploiter un parallélisme haute performance, la gestion de la dynamicité et supportent la scalabilité des applications ? Cette thèse contribue à y répondre en proposant un paradigme de programmation sur la grille [AP][APS][PACb][PACa]. Nous discuterons les limitations des modèles existants ainsi que les nouvelles possibilités et les exigences de la grille.

2.3.1 Modèles et technologies de programmation

Les vingt dernières années de recherche et développement dans le domaine du parallélisme et de la programmation et conception des systèmes distribués ont produit d'énormes avancées technologiques, guidées tout d'abord par des architectures matérielles plus efficaces (et plus pratiques) et par le désir de construire des systèmes faciles à maintenir et réutilisables. Nous donnerons un aperçu de ces modèles, langages et environements, qui pour la plupart, ont leurs racines dans le parallélisme et l'informatique répartie ordinaire et sont appliqués aux grilles parce qu'ils sont des méthodologies de programmation bien établies.

2.3.1.1 Modèles à passage de messages

Les modèles à passage de messages fournissent des abstractions de transmission de messages permettant à des processus s'exécutant sur des espaces d'adressage disjoints de communiquer. MPI (Massage Passing Interface) [SOHL+96] et PVM (Parallel Virtual Machine) [GBD+94] sont des exemples de bibliothèques à passage de messages qui fournissent des ensembles de routines de communication pour langages séquentiels tels que C ou Fortran. Ils permettent aussi la création et l'exploitation de machines virtuelles dans un système de fichier commun et suppose la fiabilité du système et des communications.

MPI est la norme la plus largement répandu en passage de messages et plusieurs implementations de cette norme existent. Parmis les plus connues[2] on peut citer MPICH [GD96] et LAM-MPI [BDV94]. Ces implémentations ne sont pas tolérante aux fautes, ne supportent pas la dynamicité, ne sont pas sécurisées et nécessitent

[2]Une liste assez complète des implémentations existantes peut être consultée sur : http ://www.lam-mpi.org/mpi/implementations/fulllist.php

un système de fichier commun. Ils sont donc adaptées aux environnements stables. Les implantations les plus connues pour la grille sont MPICH-G2 ou MPICH-V. MPICH-G2 [KTF03][AS01] est une implémentation de MPI au dessus de Globus. MPICH-V2 [BCH+] est une implémentation de MPI tolérante aux fautes (développé à l'université de Paris-XI dans le cadre du projet CGP2P de l'ACI-GRID). Il existe d'autres implémentations tolérantes aux fautes similaires comme FT-MPI [FD] ou MPI/FT [BSC+]). On peut aussi citer la bibliothèque MagPIe [KHB+99] qui propose des optimisations des opérations collectives de communication pour programmes MPI sur grille de calcul. D'autres variantes de PVM et MPI existent [PB][LT] et abordent notamment les problèmes de l'hétérogénéité et la dynamicité.

2.3.1.2 Modèles RMI et RPC

RPC (Remote Procedure Call) et RMI (Remote Method Invocation) sont deux modèles d'appels à distance qui proposent de structurer l'interaction dans l'environnement distribué davantage comme une construction d'un langage que d'un appel à une fonction de bibliothèque qui fait simplement le transfert de données entre expéditeur et récepteur. Ces deux modèles fournissent un mécanisme simple de gestion et de contrôle et vérification des calculs, données et arguments à distance. RPC et RMI peuvent être employés pour établir des modèles de plus haut niveau (à composants par exemple).

GridRPC [SNM+] est un exemple d'API RPC pour environnement Grid. En plus de fournir la sémantique standard des RPC avec exécution de tâches parallèle asynchrone, il fournit des abstractions de haut niveau par lesquels les détails de l'interaction avec la grille sont cachés. Il pourrait aussi fournir de manière transparente des capacités de gestion de ressources et ordonnancement, de sécurité et de tolérance aux fautes. La gestion des interfaces est aussi très importante dans le modèle RPC. Typiquement on utilise un langage IDL. À cet égard, GridRPC a été notamment conçu avec un certain nombre de propriétés afin d'améliorer l'utilisabilité et faciliter l'implémentation et le déploiement. Des implémentations de GridRPC au-dessus de Ninf [NSS99] et Netsolve [SYAD][Aou02] existent. D'autres mécanismes RPC pour la grille sont possible, tels que SOAP[3] et XML-RPC[4]. D'un autre coté, pour le modèle orienté objet du RPC, on peut citer Java RMI, le mécanisme d'invocation de méthodes d'objets distant de Java. Java RMI permet de créer des applications Java réparties dans lesquelles les méthodes d'objets à distance peuvent être appelées à partir d'autres machines virtuelles, notamment via le réseau. Il fournit donc une interface de programmation de haut niveau à priori bien adaptée pour le Grid [GvLPF].

[3] Simple Object Access Protocol (SOAP) 1.2. http ://www.w3.org/TR/soap12-part1/
[4] http ://www.xml-rpc.com/

2.3.1.3 Modèles à objets distribués

À la différence des modèles décrits au-dessus et qui traitent essentiellement des aspects communication, les modèles orientés objet fournissent un support plus complet pour la parallélisation/distribution d'applications. CORBA[5] (spécifié par l'Object Management Group) est un standard de modèle à objets répartis. Il permet notamment des interactions sécurisées (basées sur les RPCs, invocations de méthodes et notification d'événement) entre objets distribués et hétérogènes. L'interopérabilité et l'accès à un objet sont définis en utilisant des interfaces décrits en IDL, langage de définition d'interface, et à travers un ORB (Object Request Broker - courtier de requêtes objet), et des protocoles d'interopérabilité ; GIOP (General Inter-ORB Protocol) et sa version TCP/IP, IIOP (Internet Inter-ORB Protocol). Tandis qu'on peut le considérer comme un middleware, le but initial de CORBA est la gestion d'interfaces entre objets, c'est-à-dire les interactions client-serveur dans un environnement relativement statique. Les interactions entres objets sont étroitement couplés et le modèle assume la connaissance au moment de la compilation de la syntaxe et de la sémantique des interfaces et interactions exigées par les applications. Il fournit néanmoins un support limité à la dynamicité via l'invocation dynamique [DSI/DII (Dynamic Skeleton Interface/Dynamic Invocation Interface)] et permet une certaine personnalisation au déploiement.

Plusieurs travaux ont été mené pour améliorer les performance de CORBA sur les réseaux hautes performance ou prennant en compte le parallélisme [Den03] [DPPR]. Il y a eu aussi des efforts de recherche dans le but de rendre des service CORBA disponible sur les grilles. CoG Kit [VGLP] fournit par exemple une couche interface qui permet de faire correspondre les service de Globus et l'API CORBA.

2.3.1.4 Modèles à composants

Les composants étendent le paradigme orienté objet en permettant à des objets de contrôler les interfaces qu'ils présentent et de *composer* avec ceux présentés par d'autres. Il n'existe pas une définition consensuelle de ce qu'est un composant. La plus souvent citée est : "un composant logiciel est une unité binaire de composition ayant des interfaces spécifiées de façon contractuelle et qui définit explicitement ses dépendances de contexte. Un composant peut être déployé de manière indépendante et est sujet à composition par des tiers" [Cer04].

Plusieurs systèmes et modèles à composants ont été définit. CCM (CORBA Component Model) [WSO01] est le modèle à composant de CORBA (définit à partir de la version 3.0 dans l'architecture de CORBA). Il ajoute une couche au dessus de CORBA et permet l'instantiation dynamique et la personnalisation de composants lors de l'exécution. Cependant, il hérite de certaines des limitations de CORBA sur

[5] The Object Management Group. http ://www.omg.org/technology/corba/corba3releaseinfo.htm

la connaissance préalable au sujet des interfaces et des interactions. EJB[6] est un modèle à composant de Sun Microsystems. Il définit un standard JavaBeans qui facilite le développement, la réutilisation et l'interopérabilité de composants Java. le modèle CCA (Common Component Architecture) [AAW+02] vise particulièrement les applications scientifiques. Ce modèle traite principalement l'hétérogénéité et la séparation de l'interface et de l'exécution et emploie des appels fonctionnels pour les interactions inter-composant. Ce modèle ne permet pas la personnalisation de composants à l'exécution mais permet de les remplacer. Outre, il n'est pas tolérant aux fautes ni sécurisé. On peut aussi citer COM/DCOM de Microsoft ou Jini[7] .

2.3.1.5 Modèles orientés services

Dans les modèles orientés services tels que les Web Sevices [Dan03] et dans les spécifications OGSA/WSRF (Open Grid Services Architecture/Web Service Resource Framework) [CFF+04a][CFF+04b], la grille fournit un ensemble extensible de services (accès aux applications) avec interfaces et protocoles standardisés. Les services web sont décrits par des interfaces basées sur XML. Le standard d'OGSA/ WSRF est WSDL [(Web Services Description Language), c'est en quelque sorte l'équivalent du langage IDL pour CORBA par exemple]. L'échange de messages s'appuie sur le protocole SOAP (Simple Object Access Protocol). Un appel de service SOAP est un flux ASCII encadré dans du XML et transporté par HTTP. La publication et l'exploration s'appuient sur UDDI (Universal Description, Discovery and Integration), qui normalise une solution d'annuaire distribué de services web. Ainsi, dans une infrastructure orientée service, une application va pouvoir rechercher, assembler et coordonner les ressources nécessaires à son exécution via la publication, la découverte et l'invocation de services en utilisant ces protocoles.

Plusieurs initiatives de normalisation sont en cours initiées par les consortiums OASIS, RosettaNet et W3C : pour la définition, la publication, la découverte et invocation des services, ou la sémantique des messages, l'orchestration et gestion de sécurité des transactions, etc. Cela inclut ebXML [LFL04], GSFL (Grid Services Flow Language) [PKL], WSFL (Web Services Flow Language) [Ley01], XLang [Tha01], XACML (eXtensible Access Control Markup Language) [GM03], etc.

2.3.1.6 Modèles hybrides

Les modèles introduits jusqu'ici présentent les paradigmes de programmations parallèles et répartis les plus classiques ainsi que certaines extensions dédié au calcul sur les grilles. La grande répartition géographique inhérente à ces systèmes amène de nombreuses problématiques à gérer : l'hétérogénéité, l'interopérabilité, la sécurité, la fiabilité, la tolérance aux fautes, etc. Certaines infrastructures logicielles basées

[6]Enterprise JavaBeans Technology. http ://java.sun.com/products/ejb/
[7]The Community Resource for Jini Technology. http ://www.jini.org.

sur les modèles présentées plus haut essaient de faciliter la gestion de cette complexité en permettant de cacher l'hétérogénéité ou la répartition géographique, ou de gérer les fautes du système. Par ailleurs, ces modèles utilisent un seul paradigme de communication, à savoir le passage de messages ou l'invocation de fonction ou méthode à distance. Les modèles hybrides introduisent plusieurs paradigmes de communication, en associant généralement la mémoire partagée aux deux paradigmes de communication précédents.

Ainsi, afin de permettre d'utiliser au mieux les ressources disponibles sur la grille, ces modèles hybrides ont été proposés. En effet, des applications voudraient peut être fonctionner en multithreadé dans un espace d'adressage partagé tout en se synchronisant et échangeant des données. Un modèle associant OpenMP [DM98] (bibliothèque de parallélisation multi-tâches sur machines parallèles à mémoire partagée) et MPI a été considéré par plusieurs études [SB01][Hen00]. Cependant, une question importante dans la conception de ce type de programmes est posée : "MPI au dessus de OpenMP, ou OpenMP au dessus de MPI ?". La première approche peut nécessiter plus de synchronisation et limiter la quantité de parallélisme que OpenMP peut produire, et la deuxième exige que MPI soit réentrant (ou "*thread-safe*", supporté par MPI-2 [GLT99]) ou une gestion d'accès explicite à la bibliothèque MPI[8]. Quelle approche est meilleure ? Cela dépend finalement de l'application.

Une autre approche hybride est OmniRPC [SHTS] (utilisé en parti dans ce travail). OmniRPC a été spécifiquement conçu comme une API réentrante d'appels RPC pour clusters et grilles de calcul. OmniRPC emploie OpenMP pour le contrôle de l'exécution de threads parallèle. Ce système est déployé par notre équipe dans le cadre de collaboration avec l'équipe du Pr. Mitsuhisa Sato de l'université de Tsukuba. Un autre exemple de cette approche est MPJ (Message Passing Java) [CGJ+00]. MPJ fournit le *multithreading*, RMI et le passage de massage pour le développement d'applications (il suit la spécification MPI-1). Cette approche présente cependant des problèmes d'implémentation ; "MPJ au dessus de MPI ou une implémentation native de MPJ en Java ?". La première approche donne de bonnes performances mais pose des problèmes de sécurité et ne supporte pas les applets, et la deuxième approche souffre d'un problème de performance. Un support additionnel à la compilation pourra rendre ce modèle plus praticable.

2.3.1.7 Modèles pair-à-pair

Le modèle pair-à-pair tente de tirer profit de toutes les ressources inutilisée au sein des réseaux et suppose des échanges directs dans le système. Dans ce modèle, chaque ressource est potentiellement mise à disposition de l'ensemble des participants, chaque pair dans le système peut agir à la fois comme client et serveur.

[8] À noter que la plupart des implémentations de MPI actuelles sont basées sur la version 1.2 ; MPI 2 peine à se généraliser.

Plusieurs modèles pair-à-pair ont été proposés, plus ou moins décentralisés selon que le système de mise en relation est assuré par un index central ou délocalisés sur l'ensemble des pairs. Ce paradigme est très attrayant par la promesse de scalabilité et d'architectures très extensibles et potentiellement mieux tolérantes aux fautes car moins centralisée.

Une famille de protocoles spécifiquement conçu pour le calcul pair-à-pair est JXTA [VNRS] [TAA+]. JXTA est un ensemble de protocoles pair-à-pair "open source" largement répandu, définis comme des messages XML, et qui permettent de faire communiquer et collaborer n'importe quels dispositifs reliés au réseau. En utilisant JXTA, les pairs peuvent s'organiser et se configurer en groupes indépendamment de leur position dans le réseau et sans gestion centralisée. Ils peuvent aussi annoncer leurs ressources et acquérir les ressources fournies par d'autres. Les pairs contribuent à acheminer les messages permettant la pleine connectivité du réseau. JXTA permet de cacher les topologies complexes et dynamiques des réseaux pair-à-pair. Ces dispositifs ont fait de JXTA un modèle d'implémentation de services Grid et d'applications pair-à-pair [RGS+].

2.3.2 Projets de Grid computing

Cette section présente différents projets de grille à travers le monde. Ces systèmes visent à globaliser l'accès aux ressources informatiques et aux données. Comme discuté plus haut dans ce chapitre, cette ambition est rendue possible par une grande disponibilité des ressources de calcul et de stockage à travers le monde, accompagné d'un besoin réel en performance et puissance de calcul. La banalisation des réseaux hauts débits, l'omniprésence d'Internet et le travail de normalisation de protocoles et services (conduit notamment par le Global Grid Forum) ont aussi fortement contribué à ce travail de mutualisation.

Les premières réalisations de Grid visait à assembler des supercalculateurs interconnectés par des réseaux haute performance. Comme nous l'avons déjà mentionné, les premières expériences de ce genre ont été mené sur les noeuds d'ARPANET dès le début des années 1970. En 1995, les États-Unis lancent le réseau expérimental I-Way (Information Wide Area Year), reliant une vingtaine de centres de calcul par des liens ATM [DFP+96]. Une infrastructure appelé I-Soft [FGN+] a été mise au point pour le lancement d'applications sur cette plateforme. La bibliothèque de communication Nexus [FKT] a été adapté à l'exécution dans l'environnement I-Way et d'autres bibliothèques, dont CAVEcomm et MPICH, ont été étendu pour l'utilisation des mécanismes de Nexus. On utilisait alors le mot *metacomputing* pour désigner ces systèmes.

Les grilles ont par la suite étendu leur contexte et domaine d'intervention dans le but de devenir des "organisations virtuelles dynamiques et multi-institutionnelle"

[Fos01]. Dès le milieu des années 1990, plusieurs systèmes de calcul basés sur le Web, comme Charlotte [BKKW96], Bayanihan [Sar01], Javelin [CCI+97], SuperWeb [AISS], etc, proposent des plateformes d'exécution exploitant des ressources volontaires sur Internet. La plupart sont basés sur Java. Le déploiement se fait via un navigateur Web pour le lancement d'un appelet de calcul. Ce modèle est très limité, notamment parce qu'il ne permet pas l'accès aux systèmes de fichier locaux. La performance de la machine virtuelle Java par rapport à une exécution native est aussi mise en cause. Parallèlement, il y a eu le développement de projets de calcul global orienté application tels que GIMPS (Great Internet Mersenne Prime Search)[9] et Distributed.net, et en 1999, le projet SETI@home [ACK+]. Aujourd'hui, SETI@home a plus d'un million de participants réguliers à travers le monde, qui fournissent un taux de calcul de plus de 70 TeraFLOPS. Le potentiel de calcul est cependant beaucoup plus grand, car le nombre de PCs connectés à Internet croît rapidement et atteindra le milliard en 2015. Ces ressources connectées ensemble pourraient fournir plusieurs PetaFLOPS de puissance de calcul.

Ces projets ont ouvert la voie à beaucoup d'autres systèmes reposants sur la contribution de volontaires. Il existe aujourd'hui plus d'une vingtaine de projets jumeaux de SETI@home et qui sont à disposition d'éventuels volontaires. Dans ces systèmes, le client offre ses ressources et fait des calculs mais ne peut pas en soumettre. Il y a donc un besoin d'environnements et d'outils de développement d'applications sur les grilles. Il existe des initiatives de Grid commerciale (témoin d'une certaine maturité), parmis les plus connus on peut citer Entropia [CCEB03], United Devices ou GridXpert qui offrent des produits comme DCGrid, Grid MP et GX Synergy. Dans le milieu académique, plusieurs environnements existent, avec plus ou moins de fonctionnalités et des hiérarchies et architectures différentes. Cela inclut BOINC, Globus, XtremWeb, Netsolve, Ninf, OmniRPC, etc. La suite de cette section présente une introduction à ces environnements. Le système XtremWeb sera décrit en détail dans le chapitre 4.

2.3.2.1 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [And] est une plateforme logicielle générique pour le calcul distribué basé sur l'utilisation de ressources de participants volontaires sur Internet (*public-resource computing*). Chaque projet est indépendant et met en place ses propres serveurs et base de données. Plusieurs projets, dont Seti@home ou LHC@home du CERN, utilisent ce software pour la répartition de leur calcul.

BOINC fournit un cadre flexible et des mécanismes qui simplifient la création et distribution d'applications natives avec peu ou pas de modifications. BOINC est sécurisé et met en place un système d'authentification entre serveurs et partici-

[9]The Great Internet Mersenne Prime Search. http ://www.mersenne.org/.

pants. Les projets peuvent aussi avoir plusieurs serveurs (de données ou d'ordonnancement). La tolérance aux fautes est prise en compte par redondance. BOINC supporte aussi les applications avec de grandes masses de données (en entrée ou sortie), ou qui ont besoin de grandes quantités de mémoire. La distribution et collecte de données sont alors partagés par plusieurs serveurs. Les utilisateurs peuvent aussi indiquer les limites d'utilisation du disque et de la largeur de bande réseau, et les travaux ne sont envoyés qu'aux serveurs capables de les manipuler. Finalement, BOINC fournit une interface web d'administration (création de compte, édition des préférences, affichage de statut...).

2.3.2.2 Globus

Globus [Fa03] fournit une infrastructure logicielle qui permet aux applications de voir les ressources distribuées comme une machine virtuelle unique. Le projet Globus est un effort multi-institutionnel américain de recherche sur les grilles de calcul. C'est aujourd'hui une des infrastructures les plus utilisées dans les projets de Grid.

Globus consiste en un ensemble de composants qui définissent les services de base et les fonctionnalités requises pour construire une grille informatique[10]. Cela inclut la sécurité, localisation et gestion des ressources, gestion des données, la réservation et les communications. Le développeur d'outils spécifiques ou d'applications peut choisir dans l'ensemble des modules de Globus pour ces propres besoins. Globus est construit comme une architecture en couche dans laquelle les service de haut niveau peuvent être développés en utilisant les services de bas niveau.

Les informations sur les ressources et le statut sont fournies par l'intermédiare d'annuaire réseau basé sur LDAP (Lightweight Directory Access Protocol) appelé MDS (Metacomputing Directory Services). MDS consiste en deux composants, GIIS (Grid Index Information Service) et GRIS (Grid Resource Information Service). GRIS implémente une interface uniforme pour questionner les fournisseurs de ressources de la grille sur leur configuration, capacités et statut courants. Une hiérarchie de serveurs GIIS collectent les informations fournies par les GRIS et les intègre en une seule base de données et gère sa cohérence. Les fournisseurs d'informations sur les ressources utilisent un modèle *push* pour mettre à jour les GRIS. MDS suit donc les deux modèles (*push* et *pull*) pour la diffusion d'information. Globus fournit des composants d'ordonnancement mais n'assure pas de politiques d'ordonnancement de haut niveau, plusieurs systèmes d'ordonnancement global ont été développés en utilisant les services de Globus tels que Condor/G, AppeLeS ou Nimrod-G. GRAM (Grid Resource Allocation Manager) est le module qui fournit la gestion des res-

[10]The Globus Alliance a aujourd'hui une très grande influence dans le développement (notamment en matière de normes) dans le domaine des grilles informatique (avec des organismes comme le Global Grid Forum).

sources, de l'exécution et de son statut, et DUROC (Dynamically-Updated Request Online Coallocator) est le co-allocateur qui réalise des allocations groupées vers plusieurs GRAM. GRAM utilise GASS (Global Access to Secondary Storage) pour le transfert des fichiers de sortie des serveurs aux clients via une API utilisable depuis les applications. Globus utilise aussi GridFTP pour le transfert de données entre les noeuds de la grille. GridFTP peut se référer à un protocole, un serveur ou un ensemble d'outils [Fa03]. Tous les services de Globus cités sont construits sur l'infrastructure de sécurité GSI (Grid Security Infrastructure) qui fournit, entre autres, des fonctions d'authentification/autorisation et de confidentialité des communications.

À partir de la version 3, Globus est basé sur l'architecture OGSA (Open Grid Service Architecture), conforme aux recommandations OGSI. Il s'agit d'une encapsulation des services Globus dans des Grid Services (variantes de Web Services) qui rend l'accès à distance plus homogène et répondant à un standard.

2.3.2.3 Legion

Legion [NHG01] est un "méta-système" de grille orienté objet développé à l'université de Virginie aux États-Unis. Initié dès 1993 comme une plateforme d'exécution des programmes *Mentat* (extension parallèle de C++), le projet a été étendu et a donné lieu à une première version publique en 1997. Legion est toujours actif comme projet universitaire, mais a également suscité la naissance d'un produit commercial exploité par *Avaki Corporation*.

Legion fournit une infrastructure logicielle permettant l'accès à une machine virtuelle uniforme et cohérente. Il est organisé hiérarchiquement en classes et méta-classes. Les objets de Legion représentent tous les composants de la grille (fichiers, utilisateurs, processus, machines, ordonnanceurs, connexions...) et il gère un espace de nommage global fournissant une image unique des ressources depuis n'importe quel point de la grille. L'architecture de gestion de ressource de Legion est hiérarchique avec des politiques d'ordonnancement décentralisées et extensibles. Des ordonnanceurs comme Nimrod-G ou AppLeS peuvent être utilisés pour mettre en place des politiques utilisateurs. Legion fournit des services de bas niveau (stockage persistant, gestion de processus et de communication, sécurité...), et des services de haut niveau spécifiques aux grilles (accès aux données à distantes, support de l'hétérogénéité, etc). Legion fournit aussi un support multi-langage ; PVM, MPI, C, C++, FORTRAN, Java et l'IDL CORBA. Ces services sont fournit dans un environnement complètement intégré contrairement aux services indépendants de Globus.

2.3.2.4 Netsolve

NetSolve (GridSolve) [SYAD] est un environnement client/agent/serveur, basé sur les RPC, permettant un accès aux ressources informatiques, matériel ou logiciel, distribuées à travers le réseau. Grâce à une variété d'interfaces (Fortran, C, Matlab,

Mathematica et Octave), l'utilisateur peut exécuter des tâches de calcul sur des serveurs distants sans avoir de ressources de calcul installées sur sa machine. NetSolve est tolérant aux fautes et utilise une politique d'équilibrage de charge assurant une bonne exécution et permettant au système d'employer les ressources informatiques disponibles aussi efficacement que possible.

NetSolve a un fonctionnement très simple. Le programme réside dans le serveur, le client envoie ses données au serveur, via un agent, et les programmes (utilisant éventuellement des librairies mathématiques non-libres) s'exécutent sur ces données. Le résultat est ensuite renvoyé au client. Les serveurs peuvent être des stations de travail puissantes ou des systèmes parallèles (réseaux de stations de travail, MPP...). Les agents tiennent des traces des ressources logiciels disponibles sur chaque serveur, et à chaque requête d'un client, l'agent impliqué (et en consultant les autres) dresse une liste classés des serveurs (du plus approprié au moins approprié) qu'il fournit ensuite au client (qui contacte les serveurs de la liste pour le calcul). La structure de NetSolve contient un langage de description utilisé pour décrire les fonctionnalités numériques des serveurs. La description contient les informations sur les fonctions de la bibliothèque numérique, et les entrées et sorties. Il est donc possible de générer des fichiers de description pour n'importe quelle bibliothèque numérique.

2.3.2.5 Ninf/Ninf-G

Ninf [NSS99] est une infrastructure client-serveur de calcul global similaire à NetSolve. Elle permet l'accès aux serveurs de calcul et bases de données distants. Le client Ninf accède aux ressources en utilisant la bibliothèque client (API) de Ninf via des langages tels que C ou Fortran. Les appels de la bibliothèque peuvent être synchrones ou asynchrones. Les composants clés de Ninf sont sa bibliothèque client, le "méta-serveur" (qui maintient les informations sur les serveurs du réseau en utilisant un annuaire LDAP), et les bibliothèques distantes. Le méta-serveur alloue les ressources sur les serveurs appropriés selon des politiques d'équilibrage de charge et d'ordonnancement. Les ressources de Ninf (serveurs) enregistrent les détails des services de bibliothèques disponibles en utilisant un modèle *push*. Le modèle d'organisation de Ninf est plat. Ninf-G est une implémentation de Ninf au-dessus de Globus. Elle fournit une API GridRPC [SNM+] qui est discutée pour standardisation au *Grid Remote Procedure Call Working Group* et au *Global Grid Forum*.

2.3.2.6 OmniRPC

OmniRPC [SHTS] est un environnement de programmation par appels RPC sur cluster de PCs ou en grilles de calcul. Il suit un modèle classique client-serveur. OmniRPC a hérité son API de Ninf et utilise un modèle hybride de programmation car il emploie OpenMP pour le contrôle de l'exécution de *threads* parallèles. OmniRPC fournit aussi un mécanisme de persistance, pour exécutables et données,

via la gestion de *handles*. Comme support d'exécution, OmniRPC utilise `rsh` pour les environnements locaux, ou Globus ou `ssh` pour les noeuds distants. En outre, OmniRPC supporte les système de batch PBS et SGE.

2.3.2.7 DIET

L'environnement DIET (Distributed Interactive Engineering Toolbox) [CD05] [Sut02] est basé sur le même modèle d'appels RPC utilisé dans les environnements Netsolve ou OmniRPC mais propose en plus une approche hiérarchique des serveurs de calcul. Il utilise une hiérarchie d'agents extensible suivant la topologie du réseau qui permet de réduire le temps de propagation des requêtes depuis le client et d'éviter les goulots d'étranglement et permettre ainsi le passage à l'échelle. En effet, dans les systèmes Netsolve ou OmniRPC l'agent ordonnanceur est centralisé et peut devenir un goulot d'étranglement dans des situations réelles où beaucoup de clients souhaitent accéder à plusieurs serveurs. DIET est basé sur CORBA.

La hiérarchie de DIET est composée de deux types de composants : les agents maîtres et les agents locaux. Les agents maître sont au sommet de la hiérarchie et sont connectés par un réseau complet. Chacun des agents maîtres est la racine d'un arbre d'agents locaux (les feuilles de l'arbre sont les serveurs de calcul). DIET met en place un module de prédiction de performances basé sur FAST (Fast Agent's System Timer) [Qui02]. Cet outil permet de mettre en correspondance les mesures des disponibilités dynamiques des différentes ressources de calcul et de communication pour prédire le temps nécessaire à l'exécution d'une tâche donnée sur une machine donnée à un instant donné. Le projet de site d'expertise en algèbre linéaire creuse GRID-TLSE [DGH+04][CDL+05] utilise ce middleware.

2.3.2.8 Normalisation (OGSA/WSRF)

Les 'normes' ou 'standards' sont un des domaines de convergence des intérêts des scientifiques et utilisateurs. OGSA (Open Grid Services Architecture) développé par GGF (Global Grid Forum) vise à définir une architecture commune, standard et ouverte pour les applications sur les grilles. Le but de l'OGSA est de normaliser les services de base qui se trouvent généralement dans une application en grille (gestion de tâches, gestion de ressources, sécurité...) en spécifiant un ensemble d'interfaces standards pour ces services. Les Web Services ont été choisis comme technologie sous-jacente. Ainsi, les spécifications de WSRF (Web Services Resource Framework), développées par OASIS et qui remplace OGSI (Open Grid Service Infrastructure), fournissent une infrastructure à l'architecture OGSA. En se basant sur les Web Services, WSRF définit l'ensemble des mécanismes de gestion d'informations entre entités à la base des services OGSA, appelé Grid Services. Globus 4 suit les recommandations de l'OGSA/WSRF.

2.3.3 Discussion

Il y a actuellement un grand nombre de projets en grilles informatiques, et le domaine est en plein essor avec l'émergence de diverses approches et solutions de développements de ces systèmes. Nous avons vu qu'il existe une grande diversité d'environnements avec des spécificités différentes même s'ils sont essentiellement destinées à assurer le déploiement des applications à grande échelle (en interagissant éventuellement avec les gestionnaires de tâches classiques au niveau local). Ces environnements maintiennent donc une liste des ressources, un suivi de leur état, et la possibilité de lancer et gérer des tâches avec transferts des données et codes. Cependant la structure interne des applications n'est pas prise en charge. Ces plateformes ne fournissent aucun modèle de programmation évolué et la plupart n'intègrent pas d'infrastructure d'exécution spécifiquement adapté aux grilles. À noter aussi que ces systèmes se concentrent sur le calcul, les grilles dédiées aux traitements de masses de données (stockage, réplication, distributions...) sont très peu nombreux.

Au niveau applicatif, nous avons vu qu'une des approches consiste à voir la grille comme un calculateur parallèle, avec l'utilisation du passage de message. Cette approche ne convient pas forcément aux applications sur la grille, notamment à cause de la grande variété des ressources (la latence introduite par l'usage d'un WAN peut pénaliser lourdement les applications qui échangent de nombreux messages). En pratique, les applications parallèles exigent une certaine adaptation pour une exécution efficace sur une grille, comme la gestion de la topologie du réseau en préférant les liens rapides, la gestion de l'hétérogénéité des performances des différents noeuds et la tolérance aux fautes. Une plateforme de communication qui offre ces services à un niveau générique est aussi souhaitable.

L'approche qui consiste à voir la grille comme un système réparti classique (comme en LAN) pose aussi les mêmes problèmes. Outre, par rapport au LAN, l'utilisation à grande échelle des infrastructures réparties présente des soucis de sécurité, de connectivité et de performance. Il serait par exemple très avantageux de proposer la compression de données. Aussi, les travaux visant à porter ces infrastructures, notamment CORBA ou Java RMI, sur les réseaux hautes performance (type Myrinet ou SCI, qui font probablement partie d'une grille) restent insuffisants et il n'y a pas de généralisation de fonctionnalités vers des versions officielles. Les modèles de programmation conviennent cependant mieux à l'exploitation des ressources mais sont peu adaptés au calcul scientifique. Il n'est pas possible d'exploiter efficacement la grille pour du code scientifique avec seulement le multi-paramètres ou le client-serveur comme paradigmes de calcul.

Chacune de ces deux approches est restrictive et couvre uniquement une partie des possibilités que peuvent apporter les grilles. De par leur nature, ces environnements rejoignent tous les aspects et problématiques du parallélisme (i.e. performance), et de la distribution (comme l'hétérogénéité ou l'interopérabilité). Un

modèle de programmation pour la grille devrait prendre en compte tous ces aspects. Les modèles orientés objets, à composants ou hybrides tentent d'y répondre en introduisant des notions de plusieurs paradigmes de communication et de programmation. Utiliser une combinaison de modèles de programmation nous semble être la meilleure façon de programmer la grille pour le calcul scientifique.

2.4 L'algèbre linéaire parallèle

Dans cette section, nous présentons les travaux relatifs à l'algèbre linéaire parallèle et distribuée, ainsi que quelques bibliothèques numériques largement utilisées en calcul scientifique.

2.4.1 Travaux relatifs

Un nombre important de travaux de recherche abordent les problèmes du parallélisme et calcul à grande échelle en algèbre linéaire. Toutefois, la plupart considèrent seulement le contexte des supercalculateurs ou clusters de PCs, utilisant des extentions parallèles de langages de programmation, des bibliothèques de communication ou des compilateurs parallélisants.

Un secteur important et très actif de recherche est le développement de bibliothèques numériques. Leur intérêt est double ; premièrement, elles offrent une interface simple pour la résolution d'une grande variétée de problèmes numériques tout en laissant une certaine liberté au développeur quant au choix des algorithmes de résolution. Ensuite, elles permettent de cacher les détails de l'architecture cible à l'utilisateur. La gestion des interactions entre le matériel (les niveaux de caches, unités parallèles...), l'utilisation de bibliothèques de bas niveau, et le choix du langage ou des options de compilation incombent au développeur de la bibliothèque. Le développement de bibliothèques parallèles est basé sur des noyaux séquentiels, notamment les BLAS et LAPACK, ou des versions optimisées, telle que ATLAS qui analyse de manière extensive l'architecture cible (notamment les niveaux et tailles des caches) et son compilateur, et génère ensuite un code source qui permettra d'obtenir des performances proche des performances crêtes.

Il existe un certain nombre de bibliothèques numériques parallèles[11]. On peut en citer PBLAS (version parallèle des BLAS) ou PLAPACK. On décrira quelques-unes dans la section suivante. En outre, les techniques dites *out-of-core* sont aussi utilisées lorsque la taille des données est trop importante pour permettre un stockage en mémoire principale. Ils mettent en oeuvre des algorithmes spécifiques pour tenir compte du stockage des données sur des supports de mémoire secondaire, typiquement les disques durs. Un prototype de gestion de données *out-of-core* est disponible dans

[11] Des listes assez complètes sont données sur les sites : http ://gams.nist.gov/ et http ://www.netlib.org/utk/people/JackDongarra/la-sw.html.

la bibliothèque parallèle ScaLAPACK (Scalable LAPACK). Dans [DD00] et [Car00] des implémentations de factorisations parallèles LU, QR et Cholesky utilisent ce prototype.

Autres travaux de recherche en algèbre linéaire parallèle considèrent une approche orientée objet comme un moyen de réutilisation du code séquentiel dans des versions parallèles. La bibliothèque LAKe (Linear Algebra Kernel) [NE01] [Aou02], développée à l'université de Versailles, est basée sur cette approche. Elle implémente les méthodes itératives de Krylov pour la résolution de problèmes de valeurs/vecteurs propres.

Ces bibliothèques (qui utilisent en majorité le passage de message comme paradigme de calcul), n'ont pas été conçues pour des ressources de calcul géographiquement distribuées ; avec différents systèmes de fichiers, volatiles, hétérogène, dynamique, etc. Elles sont souvent spécialisées, que ça soit pour l'achitecture cible ou les problèmes à résoudre. Notre contexte de travail est différent et cherche à développer un cadre de calcul pour applications matricielles denses sur les grilles informatiques en suivant un paradigme de calcul plus adapté.

2.4.2 Bibliothèques numériques parallèles

Malgré d'énormes efforts de développement de compilateurs parallélisants et de langages parallèles à extensions compatibles tels que HPF (High Performance Fortran) ou OpenMP ; le passage de messages reste le paradigme le plus utilisé pour le développement parallèle en algèbre linéaire. Les bibliothèques telles que PVM, puis le standard MPI ont contribué à l'énorme succès de ce paradigme. Dans cette section nous présentons quelques efforts de développement de bibliothèques parallèles d'algèbre linéaire, dense et creuse, souvent basées sur les versions séquentielles BLAS et LAPACK.

Les BLACS

Les BLACS (*Basic Linear Communication Subroutines*) sont une bibliothèque de communication à passage de messages dédiée à l'algèbre linéaire. Elle forme une interface qui encapsule des bibliothèques de communication de plus bas niveau comme MPI ou PVM. Le but des BLACS est de faciliter la programmation d'applications matricielles et de les rendre portables. En effet, le temps nécessaire pour implémenter efficacement des algorithmes à mémoire distribuée rend impossible la perspective de réécrire le code pour chaque architecture parallèle cible. Elle offre donc l'indépendance par rapport à l'architecture. Les BLACS sont utilisés comme couche de communication de la bibliothèque ScaLAPACK.

Les BLACS comprennent des routines de communications point à point d'envoie et réception de matrices ou blocs de matrice, des fonctions de diffusion, de réduction,

et des routines destinées à la création/destruction de contexte, de consultation de paramètres tels que le nombre de processurs, disposition de la grille, etc. Les BLACS sont implémentés en C, mais une interface d'appel Fortran existe.

Les PBLAS

Les PBLAS sont la version parallèle des BLAS. Elles utilisent les BLACS pour les communications à passage de messages et les BLAS pour les calculs. Cette version met en place des optimisations réalisants des traitements par blocs. Comme les BLAS, les PBLAS incluent les 3 niveaux de calcul ; opérations distribuées vecteur-vecteur, matrice-vecteur et matrice-matrice.

PLAPACK

PLAPACK [ABE+] est une bibliothèque pour implémentation parallèle d'algorithmes d'algèbre linéaire sur des architectures à mémoire distribuées. L'implémentation parallèle de ces algorithmes nécessite généralement des manipulations d'indices et paramètres décrivant les données, leur distribution sur les processeurs, et les communications nécessaires. Les manipulations d'indices peuvent produire des erreurs dans le code, surtout pour des algorithmes plus sophistiqués, où l'effort de codage devient très lourd. L'infrastructure PLAPACK fournit une solution à cela en proposant une interface de codage qui reflète la description normale des algorithmes séquentiels. PLAPACK utilise une approche orienté objet et des distributions par blocs. PLAPACK sert aussi de base pour l'infrastructure POOCLAPACK (Parallel Out-of-Core Linear Algebra Package) [Rei] pour l'implémentation *out-of-core* d'algorithmes d'algèbre linéaire dense.

Bibliothèques pour l'algèbre linéaire creuse

Il existe un grand nombre de bibliothèques de calcul pour l'algèbre linéaire creuse, implémentants des méthodes directes de résolution de systèmes linéaires (MUMPS, PSPASES, SuperLU...), des méthodes itératives (PETSc, BlockSolve95, HYPRE, pARMS...), ou des méthodes de résolution de problèmes de valeurs/vecteurs propres (PARPACK, LZPACK, SLEPc...).

Les problèmes matriciels creux sont largement répandues dans de larges domaines d'application, notamment en simulation, météorologie ou la recherche scientifique de base. Ces bibliothèques à passage de messages, écrits pour la plupart en C ou Fortran, offrent de très bonnes performances sur les architectures cibles. Par ailleurs, la bibliothèque PETSc [BEG+97] a été utilisée dans DistSolve [Cro03] ; une interface Web Services de résolution de systèmes d'équations creux basée sur XML-RPC.

2.5 Conclusion

Nous avons présenté dans ce chapitre une vue d'ensemble du parallélisme et des grilles informatiques visant à globaliser l'accés aux ressources et aux données, avec plusieurs contextes d'application et d'utilisation. Cela inclut les réseaux multi-institutionnels connectant des centres de calcul haute performance, éventuellement via des lignes dédiés, ou les systèmes basés sur la mutualisation de ressources publics sur Internet. Nous avons décrit aussi les modèles de calcul et infrastructures associées utilisés pour la programmation et la gestion des systèmes parallèles et répartis. Nous avons conclu à la nécessité de proposer une combinaison de différents paradigmes de programmation pour le calcul scientifique sur la grille.

Nous avons aussi présenté les travaux relatifs à l'algèbre linéaire parallèle et répartie, ainsi que les bibliothèques numériques largement utilisées en calcul scientifique. Ces bibliothèques sont souvent spécialisées, que ça soit pour l'achitecture visée ou les problèmes à résoudre. Nous avons ainsi spécifié le cadre de nos recherches et mis en évidence un certain nombre de contraintes liés à la conception algorithmique sur la grille. Le chapitre suivant présente les différents aspects de la gestion des ressources et données sur les grilles informatiques.

Chapitre 3

Gestion des ressources et des données sur la grille

3.1 Introduction

Les grilles informatiques contiennent des ressources hétérogènes et dynamiques, des systèmes et politiques de gestion locales différents et des besoins variés pour les applications distribuées ; en temps CPU, entrées/sorties, mémoire et réseau. Aussi, les utilisateurs et fournisseurs des ressources n'ont pas forcement les mêmes objectifs et stratégies. De ce fait, ces systèmes sont extrêmement complexes à gérer.

La grille peut être vu comme un certains nombre de composants à différents niveaux, allant de l'accès aux ressources aux développement d'applications pour utilisateur final. Tout d'abord il s'agit de ressources (PCs, SMPs, clusters) sous différents systèmes d'exploitation et utilisant probablement des systèmes de batch (Condor, PBS...), et des mécanismes de priorité, de login, de stockage, etc. À un niveau supérieur, un ensemble de middleware (ou intergiciels) qui offrent toute sorte de services ; tels que sécurité, gestion et allocation de ressources, ordonnancement, etc. Et finalement, un ensemble d'API et de langages de haut niveau pour utilisateurs finaux. Nous nous focalisons dans ce chapitre sur les différentes approches de gestions de ressources et de données dans les grilles informatiques et introduisons les notions d'anticipation de migration et de stockage persistant des données.

3.2 Gestion des ressources

La complexité des systèmes de Grid à grande échelle rend les approches traditionnelles de gestion des ressources non appropriées, car centralisées, ont besoin d'informations complètes sur l'état du système, et adoptent généralement des optimisations de bas niveau. L'ordonnancement dans ces plateformes est sensiblement compliqué car les caractéristiques d'uniformité, des applications et ressources, dans

les systèmes locaux ne sont plus vraies dans ces systèmes. Fondamentalement, un système de gestion doit donc découvrir et rassembler l'information sur l'état des ressources disponibles, choisir les plus appropriées, prédire les performance potentielles des tâches candidates, et soumettre les tâches selon des politiques de performance (réduction du temps d'exécution, contraintes économiques, optimisation d'utilisation des ressources, etc). De façon générique, on peut voir un système de Grid comme suit : les utilisateurs interagissent essentiellement avec une entité qui attribue les ressources, ordonnance les tâches, gère la tolérance aux fautes et récupère les résultats, notamment via des serveurs d'informations. De la même façon, la gestion des ressources est un ensemble d'abstractions globales qui devraient pouvoir manipuler diverses hypothèses d'exécution des applications, en prenant en compte le type et la hiérarchie de la grille, le modèle des ressources, et les exigences de l'utilisateur.

Nous avons présenté dans [AP] un cadre de gestion de ressources en grilles informatiques comprennant un modèle de ressource, un modèle de performance et un modèle d'application. Le modèle d'application analyse les caractéristiques de l'application cible et est basé sur le placement persistant des données et l'anticipation des migrations. Le modèle de ressource fournit une représentation de la grille, incluant les caractéristiques et disponibilités des ressources de calcul et de communication, et le modèle de performance est responsable de prédire le potentiel de performance d'une application. Dans [PA04], nous avons aussi fait l'hypothèse de multicast fiable [HP] à travers la grille pour mesurer le coût d'inversion de matrices de très grandes tailles sur différentes configurations de grilles informatiques. Nous aborderons cela en détail au chapitre 6. Aussi, nous n'avons pas abordé les politiques d'ordonnancement dans la description de ce cadre de gestion, car la corrélation entre applications matricielles et politiques d'ordonnancement et de distribution/re-distribution est très complexe et conduit souvent à des résultats de NP-complétude même pour des cas très simple d'hétérogénéité.

3.2.1 Ordonnancement

L'ordonnancement est une fonction primordiale dans la grille. Tous les environnements mentionnées dans le chapitre précédent ont des fonctionnalités d'ordonnancement (plus ou moins avancées). Certains de ces environnements ont la capacité d'intégrer d'autres outils existants, de niveau local, tels que PBS, LoadLeveler, Condor, etc. Un ordonnancement de niveau supérieur est aussi nécessaire, selon l'organisation de la grille, pour un équilibrage de charge efficace entre différents sites par exemple. GRAM de Globus [Fa03] est un exemple d'interfaçage entre haut niveau et systèmes locaux autonomes.

Il existe plusieurs organisations d'ordonnancement selon le type de la grille; centralisées, hiérarchiques et décentralisées. Un exemple d'organisation centralisée (local) est Condor, basé sur le mécanisme *ClassAd Matchmaker*. Les deux autres

organisations ont des propriétés plus appropriées pour l'ordonnancement sur les grilles. Dans une organisation hiérarchique, la décision d'ordonnancement est prise sur plusieurs niveaux. Legion et Darwin utilisent ce mode hiérarchique, plus "scalable" et tolérant aux fautes que le mode centralisé. Cependant, il doit traiter les cas de politiques dynamiques d'utilisation des ressources. L'alternative est l'organisation décentralisée qui aborde naturellement plusieurs questions importantes telles que la tolérance aux fautes, le passage à l'échelle, l'autonomie des sites et les multiples politiques d'ordonnancement. Pour cette politique, l'*overhead* de la coordination entre ordonnanceurs est le facteur déterminant de la scalabilité du système global. Les systèmes comme MOL, Ninf ou Globus utilisent une approche décentralisé.

L'évaluation de l'état est un point très important dans l'ordonnancement et se base sur le degré d'information disponible et l'organisation des communications par lesquelles cette information a été distribuée. Dans la grille, l'évaluation de l'état est souvent faite partiellement, due au coût de propagation de l'information dans les grands systèmes. L'évaluation non prédictive de l'état emploie l'information et le statut courants des ressources et des jobs et ne tient pas compte de l'historique. Les approches non prédictives emploient des heuristiques ou un modèle probabiliste de distribution (basé sur une analyse statistique). L'approche prédictive tient, elle, compte de l'information courante et l'historique telle que des exécutions précédentes d'une application afin d'estimer l'état [CJ]. Des heuristiques et techniques statistiques sont aussi employées pour cette approche [KBM02][BCD]. Parmis les systèmes de collecte d'informations les plus mûrs on peut citer NWS (Network Weather Service) [WSH99][Wol] (utilisé dans nos déploiement et décrit en détail dans le chapitre suivant) et MDS2 [ZS] de Globus. Aussi, le ré-ordonnancement des tâches, périodique ou par événements, a pour but d'optimiser l'utilisation des ressources ou toute autre métrique de la politique d'ordonnancement. Cette politique peut être fixe, comme pour AppLeS, ou extensible, permettant ainsi aux entités externes de la changer (appelé schéma *ad-hoc*), comme pour le système Legion. Dans la suite de cette section nous présentons un aperçu des systèmes de gestion de ressource en local, puis sur grilles de calcul.

3.2.1.1 Systèmes de *batch*

Le coeur de la grille est la ressource local. Partant de cette constatation, les systèmes de *batch* peuvent servir de base de solution à l'ordonnancement sur la grille, mais ne peuvent pas être une solution complète car les caractéristiques, pour les ressources, les applications et la hiérarchie ne sont pas les mêmes. Un deuxième point très important est que les systèmes d'ordonnancement des grilles devraient être en mesure de coopérer avec ces systèmes d'ordonnancement locaux. Il existe plusieurs systèmes de gestion local de tâches, tels que Condor, LoadLeveler, LSF, PBS, NQE, etc. En général, chaque application parallèle fait ses demandes de ressources dans un *script* que l'ordonnanceur interprétera. Cela inclut notamment le nombre

de noeuds requis. Le système d'ordonnancement maintient une file de tâches, selon des politiques et algorithmes différents (First-Come-First-Serve, Minimal-Request-Job-First, Shortest-Job-First, Backfill, etc) [Zhu04]. Ces systèmes diffèrent dans leur philosophie de conception et implémentation. Dans [KN94], les auteurs évaluent et comparent une douzaine de systèmes de *batch* en se basant sur les critères d'hétérogénéité, supports parallèles, caractéristiques du système de fichiers et entrées/sorties, équilibrage de charge, système de sauvegarde, etc.

3.2.1.2 AppLeS

AppLeS [BWC+03] (Application Level Scheduling) est un système d'ordonnancement au niveau application développé à l'université de San Diego en Californie. Son approche se concentre sur le développement d'agents d'ordonnancement pour chaque application tournant sur la grille. L'agent aura en charge la collecte d'informations sur le système, via le système de capteurs de NWS, la mise en place de l'ordonnancement propre à l'application et l'implantation sur la selection des ressources.

AppLeS peut interagir avec d'autres systèmes de grille tels que Globus, Legion et NetSolve. Les applications intègrent les agents AppLeS et peuvent donc s'auto-ordonnancer sur la grille. Il suit le modèle de gestion de ressource propre au système de grille utilisé. Un autre effort dans le cadre de ce projet est le développement d'APST [CBOW] (AppLeS Parameter Sweep Template) qui vise les applications avec un très grand nombre de tâches indépendantes avec un partage éventuel des données en entrée, semblable à Nimrod-G [Buy02] (décrit par la suite). On peut citer aussi les outils AppLeSeeds, qui est une bibliothèque d'utilitaires pour le développement d'applications, et APT (AppLeS Process Tracker), un ensemble d'outils permettant a AppLeS de pister et suivre le progrès des processus qu'il contrôle.

3.2.1.3 Condor/Condor-G

Condor [TTL04] est un environnement de gestion de charge de travail développé à l'université du Wisconsin. Il met en place un mécanisme de file d'attente pour les jobs, une politique d'ordonnancement et de priorité et permet le contrôle de grandes collections de station de travail, en permettant notamment d'exploiter au mieux les cycles de calcul dit perdus (vol de cycles). L'architecture de Condor suit un modèle par couches et offre des services puissants et flexibles de gestion de ressources. Condor prête une intention particulière aux propriétaires de ressources et alloue les ressources à la pool selon les conditions d'utilisation qu'ils définissent. Aussi, par ses capacités d'appel à distance, Condor préserve l'environnement original des tâches sur la machine cible même si elle ne partage pas le système de fichier avec la machine source, et/ou l'identification de l'utilisateur. Condor inclut des système de sauvegarde (*checkpointing*) et de migration de tâches (très importants pour le ré-ordonnancement). Chaque pool Condor a un collecteur, qui fournit et stocke

l'information sur les ressources et collecte les annonces de disponibilité. Les agents qui tournent sur chaque ressource annoncent leurs disponibilités et demandes de ressources au collecteur. Le *MatchMaker* de Condor interroge le collecteur pour les ressources et détermine les compatibilités entre les demandes et offres de ressources. Les requêtes de demande et offre de ressources sont décrites en langage *ClassAd* (Condor classified advertisement language) [CRLS03].

Une version Grid de Condor, Condor-G [FTF+02][TTL02], a été proposé et utilise les services de Globus (GRAM, GSI et MDS). Chaque machine source construit localement un GridManager qui contrôle les tâches locales, recherche les ressources disponibles et ordonnance les tâches sur les ressources possibles. L'authentification (pour l'inter-domaines) et la collecte d'information sont basés sur GSI et MDS. Condor-G est tolérant aux fautes et permet l'exécution inter-domaine sur des ressources à distance qui exigent l'authentification.

3.2.1.4 Nimrod-G

Nimrod-G [Buy02] est un système de gestion de ressources, de contrôle et d'orientation de tâches d'applications de type *task farming* sur la grille. Nimrod-G est un système paramétrique, qui suit un modèle hiérarchique et un modèle de marché pour la gestion de ressources (définis dans [Buy02]). Il emploie les services de systèmes de Grid tels que Globus et Legion pour la gestion bas niveau des ressources. Nimrod-G utilise un annuaire réseau ou une organisation de données basée sur un modèle objet. Nimrod-G intègre un modèle économique pour la gestion de ressource et l'ordonnancement. Cela veut dire qu'il peut ordonnancer des applications sur la base de dates limites d'exécution et du budget.

Nimrod-G permet la recherche de ressources, la négociation, l'ordonnancement, la récupération des résultats, et la présentation finale à l'utilisateur. Il emploie les services de GRACE [BAG] (Grid Architecture for Computational Economy) pour la négociation dynamique avec les agents des propriétaires des ressources pour le choix des ressources les plus appropriées. L'évaluation des capacités de la grille est effectuée selon des heuristiques et le profilage historique des charges. La politique d'ordonnancement des applications est orientée par les conditions définies par utilisateur. L'équilibrage de charge est exécuté via un ré-ordonnancement périodique.

3.2.2 Modèles économiques

L'approche économique fournit des politiques et outils de partage et allocation de ressources basés sur l'échange ou la facturation. Elle fournit une bonne base de contrôle de la décentralisation et l'hétérogénéité présentes naturellement dans le monde économique réel. La plupart des systèmes d'ordonnancement et gestion des ressources cités auparavant adoptent une stratégie classique où l'ordonnanceur décide quelle tâche doit être exécutée sur quelle ressource dans le but d'optimiser

l'utilisation de la plateforme ; les temps d'exécution ou toute autre métrique liée à l'application ou aux ressources. Ces ordonnanceurs ne prennent pas en compte le coût (prix) d'accès à la ressource et ne posent pas de conditions (date limite par exemple) sur la fin de l'application. Les modèles économiques introduisent donc la notion de compétition entre utilisateurs/propriétaires des ressources, et une évaluation basée sur l'offre et la demande, et des prix et dates limites.

Plusieurs systèmes explorent l'utilisation de différents modèles économiques pour le commerce et la gestion des ressources. On peut en citer Spawn [WHH+92], Popcorn [NLRC98], Java Market [Bor00], Enhanced MOSIX [AAB+00], OCEAN [PHP+], G-Commerce [WPBB01], Nimrod-G [Buy02], etc. Ces systèmes incluent plusieurs domaines d'application, notamment le calcul basés sur le Web (Popcorn, Java Market, JaWS) ou la gestion de grilles de calcul : sur un seul domaine (Enhanced MOSIX) ou multi-domaines (Nimrod-G). Il existe aussi des systèmes de gestion de bases de données distribuées (Mariposa [SAP+96]) ou de stockage (Stanford Peers [BH]) basés sur une approche économique.

L'approche la plus extensible semble être celle de GRACE (Grid Architecture for Computational Economy) qui sert de base d'implémentation pour Nimrod-G. Elle offre une architecture à couche capable d'intégrer les services d'infrastructures existantes (comme Globus ou Legion) pour exécution sur des grilles étendues (*wide-area*). À noter finalement que la technologie des grilles informatiques (et sa communauté d'utilisateurs) est assez récente et n'est pas encore acceptée et reconnue en commerce.

3.3 La localité des données sur la grille

L'accès aux données est une problématique très importante dans la gestion de la grille. Les performances de déploiement des applications dépendent largement des performances d'accès et de l'emplacement des données. Un des challenges est de pouvoir exécuter des calculs sur d'énormes quantités de données dans des temps raisonables. Mais où se trouvent ces données et où vont-elle après traitement ? Car le problème majeur est celui de la localisation des données au moment du calcul et par conséquent de la vitesse et du coût d'accès à ces données. Il y a aussi un problème de capacité de stockage que nous n'aborderons pas ici. Le principe de notre approche est que toute donnée nécessaire à l'exécution et qui n'est pas en train d'être calculée ailleurs, est présente sur le noeud de calcul, et que tout traitement sur des données qui se trouvent sur un noeud (ou site) particulier doit être fait sur ce même noeud (ou site). Ceci est une approche simple et réaliste d'optimisation des communications et des coûts de traitement. Plusieurs recherches ont été menées dans plusieurs projets de Grid pour traiter le problème d'accès et d'échange de données. Nous les évoquerons par la suite. Nous parlerons aussi des systèmes de fichiers pour grilles informatiques.

3.3.1 Gestion et échanges de données

La nature même des grilles, et notamment la répartition géographique, rend la gestion optimale des données très difficile. La plupart des systèmes ont recours à la réplication pour les amener au plus près des utilisateurs qui en ont besoin. Le projet DataGrid [SDL+] du CERN gère la réplication des données et la mise en cache via des modules d'optimisation de requête, de gestion des accès et de déplacement des données. Selon le type des données à traiter, leur proximité ou la topologie de la grille, il établit un plan d'action pour optimiser les temps d'accès. Le module de déplacement de données exécute ensuite ce travail. Le module d'accès aux données prend la suite pour la gestion du déplacement physique des fichiers. Ce travail s'exécute au niveau du système d'exploitation de chacun des serveurs, où chaque système de fichiers peut être spécifique. Dans certains cas, les fichiers sont référencés localement avec un nom défini dans une arborescence. Dans d'autres, le système utilise un catalogue dans lequel les données sont répertoriées, et la sélection du fichier se fait par itération à travers un ensemble d'attributs ou en utilisant un identifiant d'objet. Le projet DataTAG[1] (suite de DataGrid) implique d'autres sites de projets de Grid américains. Ces deux projets prêtent une attention particulière à la gestion de masses de données car les applications visées manipuleront des volumes de données sans précédent dans les années à venir (de l'ordre de plusieurs péta-octets), notamment pour le collisionneur/accélérateur de particules LHC dont le démarrage est prévu pour 2007.

Les projets de grilles de calcul cités auparavant utilisent (selon leur conception et architecture) des organisations et protocoles différents pour la gestion et transfert des données. Tandis que Legion par exemple considère un espace de nom commun et global à la plateforme, Globus utilise un service d'annuaire et GridFTP pour le transfert des données. Les autres systèmes sont souvent centralisés et utilisent une gestion de données élémentaire ; pas de réplication, ni de compression ou de gestion de localité. Par ailleurs, le passage à l'échelle nécessitera l'étude de protocoles efficaces de partage des données permettant notamment de réduire les *overheads* (liés aux protocoles, aux technologies sous-jacentes ou aux types des données), ou la consommation de bande passante (comme le protocole BitTorrent[2]). Nous considérons qu'une gestion efficace des données est capitale pour avoir de bonnes performances et qu'un plus grand effort devrait être fait dans ce sens là. Nous introduisons dans les sections suivantes les systèmes de fichiers en réseau et la persistance des données.

3.3.2 systèmes de fichiers

Le principe des systèmes de fichiers est un concept bien établi de stockage et d'accès aux données. Il offre une vue abstraite sur l'ensemble des données. Habi-

[1] http ://datatag.web.cern.ch/datatag/
[2] http ://www.bittorrent.com

tuellement les fichiers sont organisés en arborescence et portent un nom unique. Les opérations sur les fichiers (ouverture, lecture, écriture...) sont fournies par le système d'exploitation. Les unités de gestion des systèmes de fichiers incluent les disques, partitions, blocs, et d'autres extentions qui dépendent du système d'exploitation. Il peuvent aussi inclure la compression ou le chiffrement automatique des données, une gestion plus ou moins fine des droits d'accès aux fichiers, ou une journalisation des écritures, etc. Un système de fichiers peut être local, partagé ou en réseau. Pour les grilles de calcul, le nombre important d'interconnexions et protocoles réseau utilisés rend difficile la généralisation des approches existantes car elles obéissent généralement à certains critères de distance (longueur maximum des câbles par exemple) ou des contraintes de temps ou de taille des données, dues à leur conception.

Les grilles informatiques sont des systèmes fortement distribués. Un système de fichiers global devrait fournir un espace de nom global commun, des services avancés d'authentification, des accès transparents et indépendants de l'emplacement et un support commun à différents systèmes d'exploitation. Il doit aussi prendre en charge les fichiers volumineux et la scalabilité à des milliers d'utilisateurs et des millions de fichiers. Le GFS-WG (Grid File System Working Group) travaille sur les spécifications et architecture des services des systèmes de fichiers pour grilles informatiques. Il existe un certain nombre de systèmes de fichiers en réseau qui étendent les fonctionnalités des systèmes de fichiers distribués traditionnels tel que NFS.

AFS (Andrew File System) est un système de fichier en réseau qui permet le partage de données au travers de réseaux locaux ou à grande échelle. Il présente une architecture similaire à NFS - sans la limite liée au point de montage spécifique. DFS (Distributed File System) est une version dérivée de AFS. AFS/DFS sont basés sur les notions de cellules et volume. Les cellules sont des ensembles de serveurs et clients regroupés en unités indépendantes, et le volume est l'unité de base du stockage (regroupant un ensemble de fichiers ayant un rapport entre eux - répertoire ou point de montage de partition). AFS permet la réplication (de volumes), la tolérance aux fautes, et propose des mécanismes de mise en cache, de vérification de cohérence, et de sécurité et authentification. Toutefois, l'architecture logicielle et mise en place de ce système est assez complexe. OpenAFS[3] est la version open source de AFS.

SRB (Storage Resource Broker) est un autre outil d'accès uniforme aux données sur réseaux hétérogènes. L'architecture de SRB présente deux entités de base ; les serveurs SRB pour l'accès aux emplacements physiques des données, et les serveurs SRB-MCAT qui contiennent la liste de tous les fichiers et leurs métadonnées (droits, propriétaire...). On peut aussi citer les projets EDG/EGEE (avec RLS/LCG) [BCL+] développés au CERN. Ce sont des infrastructures de gestion de données sur la grille conçues pour les besoins du projet LCG (LHC Computing Grid).

[3] http ://www.openafs.org/

Un autre projet de gestion de données distribuées sur la grille (un peu similaire à SRB) est Mobius [LHO+]. Il consiste en un ensemble de services et protocoles qui permet la gestion et l'accès aux données distribuées (définies en XML), la création de bases de données et leur gestion en fédération. Il est définie par trois services : le GME (Global Model Exchange) responsable du stockage et de la localité, Mako (Metadata and Data Instance Management) qui définie à la fois l'interface (basée sur XML) et le protocole d'accès uniforme aux données, et DTS (Data Translation Service) qui assure la cohérence entre modèles de données des différentes institutions sur la grille.

Dans un autre contexte, le système de fichiers pair-à-pair Pastis [BPSa][BPSb] est complètement décentralisé et présente une architecture permettant le passage à l'échelle. Il utilise les infrastructures Pasty et Past comme support pour le routage, la localité et le stockage. On peut aussi citer les projets Grid Datafarm [YTS], OceanStore [KBC+] et son prototype Pond [REG+], la technologie "Storage Tank" d'IBM, et le système FarSite de Microsoft.

Ces systèmes offrent un moyen simple et transparent de gestion de la localité et le partage de données en offrant un espace de nom commun et global sur le réseau. Néanmoins, leur applicabilité à de larges grilles de calcul très étendues et les performances et fiabilité, liés souvent aux technologies sous-jacentes - de transfert, de gestion de latence, de gestion des méta-données, de mécanismes d'authentification, de cohérence et mise à jour, etc - reste à évaluer.

3.3.3 Le stockage persistant

Il existe plusieurs définitions du stockage persistant. En Java par exemple, la persistance signifie la possibilité d'accéder à un objet et son information même après la fin du programme. Un objet persistant exprime donc la survie de l'information. Pour OceanStore, le stockage persistant est la haute disponibilité, la cohérence et la continuité d'accès aux données sur une infrastructure constituée de serveurs non sûrs. La notion du stockage persistant assure donc la fiabilité et continuité de service dans certains systèmes, et la survie des données dans d'autres.

Notre définition du stockage persistant est liée aux placements des données pour les applications matricielles sur les grilles de calcul. Cette notion se rapporte à l'anticipation de migration et au clouage des données (figure 3.1). Plus le taux d'anticipation est grand, plus on satisfait le schéma du stockage persistant car le schéma du clouage est fixe pour une application donnée. Pour chaque application, le stockage persistant correspond donc à des optimisations algorithmiques de distribution et placement des données de façon à avoir le minimum de communications. Le placement des tâches de calcul s'effectue alors selon cet emplacement et le flux des données.

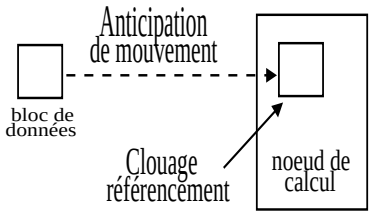


FIG. 3.1 – Opérations du stockage persistant.

Le support à la persistance peut être explicite ou implicite. Dans le premier cas, les appels et désignations explicites de stockage et récupération sont gérés par le programmeur, notamment via des *handles* ou fonctions. Le cas implicite pourrait être lié à un système de fichiers global virtualisant les accès aux données désignées. Finalement, chaque application a son schéma de stockage persistant et un effort algorithmique est requis à ce niveau. Dans nos implémentations nous avons simulé la persistance des données via la génération de blocs de matrices sur les noeuds de calcul étant donné que le middleware utilisé ne gère pas la localité des données et tâches.

3.4 Conclusion

Dans ce chapitre, nous avons vu que les applications sur la grille ne sont pas uniquement tournées vers le calcul. La gestion des données distribuées est d'une importance primordiale. Que ça soit pour la localité des données d'une application ou la gestion des espaces disques (stockage, systèmes de fichiers...) ; la répartition et distribution des données est un facteur capital de performance, mais aussi de bon fonctionnement. En effet, les besoins peuvent se chiffrer en plusieurs téra-octets (ou même en péta-octets) pour les applications scientifiques : en entrée des programmes, en communications et espaces temporaires, et en sortie. Il faudra donc adapter les algorithmes, les modèles de calcul, et les outils de déploiement et de gestion.

Pour les applications matricielles sur la grille, les supports à la persistance, à l'anticipation de migrations et gestion de références sur les données, ainsi que des protocoles optimisés de gestion de transfert, réplication et compression de données sont nécessaire. Dans le reste du document, nous parlerons plus en détails des outils utilisé ainsi que la mise en place de techniques adaptées au calcul matriciel sur la grille.

Chapitre 4

Déploiements de middlewares et d'outils expérimentaux, première expertise

Ce chapitre présente le principal middleware utilisé dans nos implémentations et le déploiement et administration des plateformes d'expérimentation. Nous décrivons aussi un aspect très important des grilles informatiques, à savoir, la modélisation et simulation. Nous finirons par présenter Grid'5000 ; la plateforme expérimentale nationale qui réunit huit Clusters répartis en France et reliés par Renater.

4.1 Introduction

Les propriétés des grilles informatiques restent mal connues et plusieurs recherches dans des domaines classiques, incluant stockage et mouvement des données, ordonnancement, protocole de communication, etc, sont nécessaires. Par rapport aux systèmes parallèles classiques, la grille possède deux caractéristiques fondamentales ; l'échelle du nombre des ressources et leurs grande complexité. D'un autre côté, le besoin de simulation/émulation et de modélisation des systèmes complexe est avéré. Le contrôle, répétition et l'étude des phénomènes d'un système de grille à l'échelle réel, pour l'évaluation des performances ou de stratégies d'ordonnancement par exemple, est tout simplement inconcevable. En effet, les performances doivent être évaluées sous différents scénarios ; changement du nombre des ressources et/ou d'utilisateurs, et avec différentes conditions. Ceci est très dur, et peut être même impossible à faire en déploiements de grille réels, d'une façon répétée et contrôlable, car la disponibilité des ressources et leurs charges varient continuellement. Il est aussi impossible de contrôler des utilisateurs et machines se trouvant dans différents domaines administratifs. Clairement, chaque déploiement est unique.

Néanmois, l'expérimentation in-situ sur des systèmes déployés est nécessaire.

Cela permet une évaluation générale des technologies existantes et la recherche et développement de techniques et solutions pour le calcul haute performance. D'ailleurs, l'étude expérimentale des grilles s'est essentiellement concentrée sur des déploiements à échelle réelle impliquant l'étude et le développement de logiciels système et de middlewares, même si cela pose naturellement des problèmes de méthodologie suivie et généralisation des résultats. Néanmois, ceci est une suite logique à l'évaluation théorique par modèles ou par simulations.

Nous présentons dans la section suivante quelques outils de simulation pour la grille, dont l'outil SimGrid que nous avons étudié au début de cette thèse. Ses fonctionnalités étaient cependant peu évoluées et ne permettent pas, par ailleurs, l'évaluation algorithmique. Nous détaillons ensuite XtremWeb, et les plateformes de calcul utilisée pour nos expérimentations. Nous parlons aussi des outils de collecte d'informations utilisés et de la version décentralisée, XtremWeb-CH. Nous présentons ensuite la plateforme expérimentale Grid'5000, et notamment le noeud d'Orsay, Grid eXplorer. Enfin, on établie une première analyse sur l'utilisation de ces outils expérimentaux.

4.2 La simulation pour la grille

Les concepteurs des systèmes de gestion des ressources et ordonnancement ou d'algorithmes pour les systèmes distribués à grande échelle ont besoin de cadres de simulation déterministe des ressources et des applications pour évaluer leurs conceptions, stratégies et algorithmes. Quand un accès à un banc d'essai n'est pas disponible, son déploiement est assez coûteux. En outre, même si le banc d'essai existe, il est limité en nombre de machines et domaines et les tests de scalabilité et d'adaptabilité des algorithmes, et l'évaluation des performances sous divers scénarios est difficile à mettre en oeuvre, à tracer et consomme beaucoup de ressources et de temps. La reproductibilité des expériences en vue de comparaison d'heuristiques d'ordonnancement est aussi impossible à garantir en cas de plateforme de calcul réelle. Les chercheurs de la communauté ont vite identifié l'importance et le besoin de tels environnements de simulation et modélisation [Buy02], et beaucoup de systèmes ont été proposés.

La simulation a été employé intensivement pour l'évaluation et modélisation des systèmes du monde réel ; de la finance et sciences de la vie, à la conception de puces informatiques. Beaucoup d'outils et technologies standards ou dédiés aux applications ont été proposés. Cela inclut des langages de simulation (comme Simprocess ou Simscript)[1], des environnements ou bibliothèques de simulation (Parsec ou SimJava), ou des simulateurs dédiés (OMNet++ pour la simulation des réseaux de communication). Il existe actuellement une grande collection de connaissances et

[1] http ://www.simprocess.com/

d'outils en simulation, néanmoins il y a peu d'outils disponibles pour les environnements Grid. Les plus connus sont : SimGrid, GridSim, MicroGrid et Bricks.

SimGrid [Casb] est un outil de simulation d'applications distribuées sur la grille. Cet outil modulaire, écrit en langage C, est spécifiquement dédié à la recherche en ordonnancement. Il fournit un ensemble d'outils pour l'ordonnancement de tâches sur les noeuds (pour les calculs) ou les liens (pour les transferts) et gère les dépendances entre tâches. Les décisions d'ordonnancement peuvent être prises par différentes entités. SimGrid implémente plusieurs modélisations des ressources et utilise les abstraction d'agent, hôte, tâche, route et canal [Leg03]. Un ordonnancement est décrit en terme d'agents communicants s'exécutant sur des hôtes et pouvant envoyer, recevoir ou exécuter des tâches. Un agent ne peut accéder directement aux ressources de calcul ou de communications mais peut les utiliser par le biais des différentes fonctions de traitement de tâches. L'envoi et réception des tâches se fait par le biais du canal (chaque hôte a sa boite aux lettres). Il suffit donc de définir le code des différents agents, créer la plateforme (hôtes, liens, routes...), et déployer les agents sur la plateforme pour lancer une simulation.

GridSim [MB][SPBT] est un outil de simulation pour la grille à événement discret écrit en Java (utilisant le package SimJava). Il permet la modélisation et simulation des entités hétérogènes de la grille ; ressources, utilisateurs, gestionnaires ou agents, ordonnanceurs et tâches d'application, ainsi que leur caractéristiques. Il peut être employé pour simuler des ordonnancements d'application pour des systèmes distribués sur un ou plusieurs domaines d'administration. Dans un environnement de grille informatique, chaque utilisateur peut avoir son propre agent ou gestionnaire, et par conséquent les conditions et objectifs d'optimisation sont différents. Ceci peut être modélisé par GridSim. Il prend en compte aussi les politiques d'optimisations globales comme dans un Cluster de PCs par exemple. Il fournit un ensemble de primitives permettant la création de tâches d'application, et de tracer et gérer les ressources. GridSim a été utilisé pour le développement du système de gestion de ressources Nimrod-G basé sur un modèle économique (de délais et budgets) [Buy02].

L'émulateur MicroGrid [LXC] fait partie du projet GrADS/VGrADS (Grid Application Development Software/Extending GrADS). Il permet l'exécution d'applications construites avec Globus dans un environnement émulé de grille virtuelle contrôlée. Il offre aux programmeurs d'applications sur la grille la possibilité d'exécuter leur application sur une grille virtuelle dont le comportement est bien mieux maîtrisé qu'une vraie grille de calcul. Il permet donc de virtualiser chaque ressource d'une grille, telle que la mémoire, le calcul, le réseau, etc. Le besoin de MicroGrid de disposer d'applications construite avec Globus impose des frais significatifs de développement car les concepteurs d'algorithmes d'ordonnancement travaillent généralement avec des modèles au lieu de construire des applications réelles.

Le système Bricks [TMN+] est l'un des premiers simulateurs proposés pour l'étude des problèmes d'ordonnancement. Il fournit une aide à la simulation de systèmes de calcul global client-serveur qui fournissent un accès à distance aux bibliothèques scientifiques et packages sur architectures parallèles. Ce système, écrit en Java, suit une méthodologie d'ordonnancement centralisée et global. On peut aussi citer deux autres simulateurs pour la grille : GangSim [DF] dédié aux problèmes d'ordonnancement et OptorSim [BCC+03] pour l'étude des stratégies de stockage et réplication de données.

La majorité des résultats obtenus avec ces systèmes sont basés sur des hypothèses parfois très restrictives ou des modèles assez simples, et toute la difficulté est de connaître leur degré de fiabilité. En effet, les réseaux d'interconnexion sont souvent simplistes en regard de la réalité des grilles, notamment en ce qui concerne la scalabilité et les heuristiques. En effet, l'hypothèse que les heuristiques prenant en compte l'hétérogénéité des ressources (de calcul et de communication) sont capables d'obtenir des prédictions parfaites des différences de performance est souvent faite. Même s'il est nécessaire de faire des hypothèses simplificatrices pour comprendre certains phénomènes et réussir à concevoir des heuristiques efficaces, le retour à la réalité est essentiel car ces hypothèses ne sont généralement pas vérifiées en pratique. Nous décrirons dans la suite de ce chapitre nos plateformes de calcul et les outils expérimentaux.

4.3 L'environnement XtremWeb

XtremWeb [LFC+03][LC03][Fed03] est un système d'expérimentation généraliste de calcul global développé en Java. Cet environnement peut être utilisé selon plusieurs contextes ; PCs volontaires sur Internet avec connexions intermittentes, et réseaux locaux ou clusters de PCs appartenants à différentes institutions ou en Intranet.

Pour correspondre aux besoins du calcul scientifique, les environnements de calcul global doivent être extensibles, sécurisés et tolérants aux fautes. XtremWeb met en place, en partie, ces fonctionnalités. Il dispose aussi d'une interface de programmation permettant le développement et l'exécution d'applications parallèles selon le paradigme maître-esclave ou multi-paramètres. Les principale caractéristiques sont une architecture client/coordinateur/worker et modèle *pull* (les workers ne sont pas sous le contrôle du coordinateur), et l'authentification et confinement d'exécution. Les section suivante donnent un peu plus de détail sur ce système.

4.3.1 Architecture et implémentation

L'architecture d'XtremWeb distingue trois rôles essentielles ; le coordinateur, le client et le worker. Le coordinateur assure la mise en relation entre les demandes de

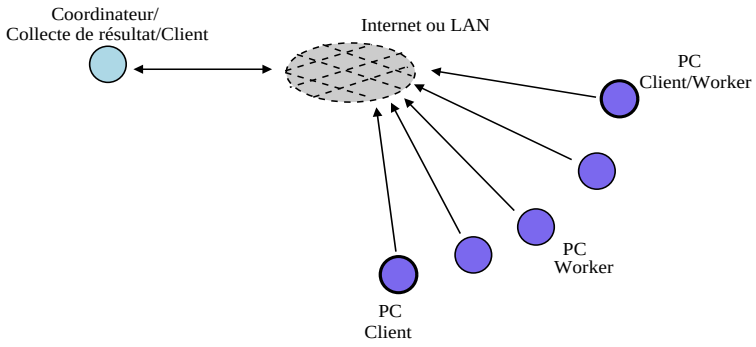


FIG. 4.1 – Organisation d'XtremWeb.

ressources provenant des clients et les ressources fournis par les workers. Il gère les tâches de calcul, la collecte des résultats et l'échange d'informations dans le système. XtremWeb a donc une architecture centralisée et suppose que le coordinateur est exécuté sur une ressource stable. Néanmoins, le contexte dans lequel XtremWeb est utilisé ici (réseaux locaux et clusters) assure la stabilité du système et une très faible fréquence des fautes, qui sont par ailleurs gérées explicitement dans les codes de nos applications.

Les versions actuelles d'XtremWeb se présentent selon l'organisation illustrée par la figure 4.1, avec un coordinateur unique. La possibilité d'utiliser plusieurs coordinateurs aurait été préférable, notamment pour éviter un goulot d'étranglement dû aux communications centralisées. La particularité d'XtremWeb par rapport à d'autres systèmes basés sur le paradigme maître-esclave est le modèle *pull* (par opposition au modèle *push*). Les workers retirent leurs travaux du coordinateur central, et ne sont pas sous son contrôle. Cette différence dans la conception d'XtremWeb s'explique par le fait qu'il vise de très grand nombre de ressources volatiles. Dans sa conception, XtremWeb vise une couche de communication RPC générique multiprotocoles construite le plus indépendamment possible des différents protocoles existants. Cette couche est décomposée en trois niveaux de communication ; protocole, transport et connexion. Le niveau protocole implémente la sérialisation des arguments, l'appel des procédures sur les ressources distantes et l'acheminement de la réponse. Trois protocoles seront supportés pour le transfert des données (Java RMI, XML-RPC et TCP). Le niveau transport assure le transport point à point. Enfin le niveau connexion est destiné aux opérations de contrôle et établissement des connexions. La sécurisation des communications entre coordinateur et workers fait appel à des techniques de cryptage de données et mécanisme d'authentification par clés publiques/privées.

Une fois le worker XtremWeb lancé, il se comporte comme n'importe quel démon (l'activité de l'utilisateur et l'usage du processeur ne sont pas pris en compte).

Une solution consiste à définir une priorité d'exécution "nice" moins élevé afin de ne pas dégrader les performances des autres travaux (si besoin est). La protection de l'intégrité du worker (dans le cas des codes natifs) est assurée par un mécanisme de confinement d'exécution (*sandboxing*) dans un environnement où les opérations systèmes sont contrôlées. Les workers s'adaptent aussi à des déconnexions temporaires, c'est-à-dire que l'utilisation des ressources réseau est indépendante de l'utilisation des ressources de calcul (le calcul off-line). L'implantation du worker s'articule autour d'une file d'attente de tâches qui est vidée et remplie par le gestionnaire de communications et l'exécuteur. Cette file contient les descriptions des tâches téléchargées et peut être sérialisée en vue d'une reprise après interruption. Le worker supporte aussi les machines multiprocesseurs via une implémentation multi-threadée de l'exécuteur.

Le coordinateur d'XtremWeb centralise la gestion des ressources présentes sur les noeuds de la plateforme. Il stocke les binaires et les tâches soumises par les clients, et en fournit aux workers. Distribuer une application consiste à fournir le binaire et une description (comprenant le nom, l'architecture, les paramètres et un fichier de configuration). Le client soumet ses tâches, via un programme client, en utilisant un script ou en ligne de commande, en fournissant la référence de l'application et l'ensemble des paramètres (arguments et/ou fichiers). La tâche est ensuite enregistrée sur la base de données MySQL et reçoit un identifiant unique. Le coordinateur fournit aussi un ensemble de pages PHP pour soumettre des tâches, suivre leurs progression, récupérer les résultats et obtenir quelques statistiques sur l'activité de la plateforme.

Le mode de placement des tâches dans XtremWeb est trivial et est basé sur une FIFO. Il n'y pas de système de priorité et la politique de placement est non configurable. Par ailleurs, la politique de choix du worker pour une tâche donnée n'est pas clairement identifiée. Cela donne parfois un comportement de soumission de tâches surprenant où des tâches indépendants sont soumis "séquentiellement" sur un même worker alors que plusieurs sont disponibles et on-line.

Les communications à destination du coordinateur sont faites à l'initiative des autres entités. Ceci facilite le déploiement au regard des pare-feux réseau qui pourraient bloquer les requêtes provenant du coordinateur situé sur un autre site. Dans l'autre sens, il est donc nécessaire de procéder à l'ouverture de ports sur le coordinateur protégé par un pare-feu. Les communications entre entités utilisent le RPC multiprotocole évoqué plus haut (actuellement implémenté sur les Java RMI). Le lecteur intéressé par plus de détails sur l'environnement XtremWeb pourra consulter la thèse de Gilles Fedak [Fed03].

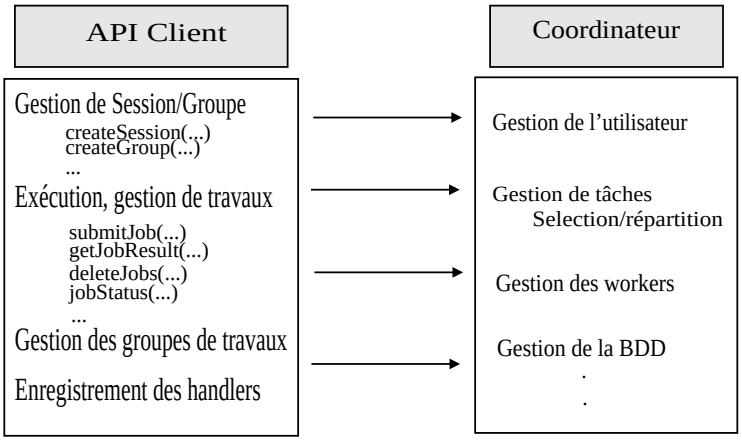


FIG. 4.2 – L'API d'XtremWeb.

4.3.2 Développement d'applications

XtremWeb offre une interface de programmation client pour le développement d'application. Cette API Java consiste en un ensemble de routines qui permettent la création et soumission, le contrôle, et la destruction de travaux. Elle permet à l'utilisateur du système de dialoguer avec le coordinateur pour spécifier son environnement d'exécution, notamment le nombre de workers, et la gestion de l'exécution. Cette dernière est géré avec les notions de sessions, groupes et contrôle par événement. L'approche de délocalisation d'exécution dans XtremWeb est la suivante ; pour soumettre un travail il faut fournir le nom de l'application et un fichier compressé qui est l'image d'un répertoire ou de fichiers qui seront injectés dans l'entrée standard du processus et les arguments de la ligne de commande de l'application. À sa réception, le worker crée un répertoire temporaire puis l'exécuteur décompresse le fichier compressé (au format zip) et lance le procesus en positionnant son répertoire de travail courant sur le répertoire temporaire. Les nouveaux fichiers créés, ainsi que les fichiers modifiés, sont compressés dans une archive qui est renvoyée au coordinateur.

L'interface de programmation se décompose en cinq parties : ouverture/fermeture de sessions, configuration de l'environnement d'exécution, gestion des travaux, gestion des groupes de travaux et enregistrement des *handlers*, i.e. les fonctions déclenchées lorsque les statuts des travaux ou groupe de travaux sont modifiés sur le coordinateur. La figure 4.2 montre quelques fonctions de l'API client d'XtremWeb.

4.3.3 Déploiements et administration

Le cadre utilisé ici est celui de réseaux locaux et clusters de PCs distribués dans de multiples domaines d'administration, en l'occurrence à l'université de Lille 1, l'université de Paris-XI à Orsay et l'université de Tsukuba au Japon. La figure 4.3

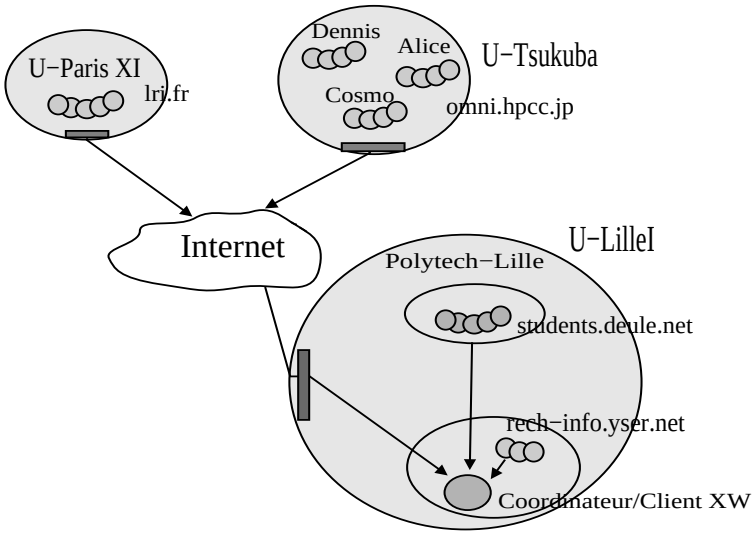


FIG. 4.3 – Configuration des plateformes d'expérimentation.

Nombre	CPU	Mémoire
28	Pentium III (Katmai), 450MHz	128MB
28	Intel Celeron, 2.2GHz	512MB
23	Intel Celeron, 2.4GHz	512MB
15	AMD Duron, 750MHz	256MB
14	Celeron (Coppermine), 600MHz	128MB
8	Intel Celeron, 2GHz	512MB
8	Intel Celeron, 1.4GHz	256MB
4	Pentium 4, 2.4GHz	512MB

TAB. 4.1 – Configuration des machines à Lille

montre le contexte d'utilisation du système XtremWeb. L'installation du coordinateur requiert la présence préalable du serveur web Apache avec le module PHP, de MySQL, et d'une JDK. La configuration du logiciel nécessite l'édition d'un fichier texte fixant les paramètres d'installation et d'administration, du coordinateur et de la base de donnée. Les options de configuration concernent essentiellement l'authentification (nom d'utilisateur et mot de passe), le nom de la base de donnée, l'attache au réseau pour les workers (dont le nom ou adresse IP du coordinateur) et la période des signaux entre worker et coordinateur. Cela devrait évoluer vers d'autres options plus avancées concernant le choix des politiques de sélection/répartition des tâches et/ou workers, le choix du ou des protocoles de transport, le choix de confiner les exécutions ou non, et la politique associée, selon qu'on utilise des ressources de confiance ou pas, etc.

Pour le déploiement et l'administration des plateformes de calcul, un ensemble de scripts shell a été écrit pour le lancement, l'arrêt et la scrutation de la plateforme, en ligne de commande ou via le "cron" pour avoir une continuité de service. Le coordinateur et le worker sont donc relancés automatiquement après un arrêt du programme ou redémarrage de la machine. L'utilisation de ces scripts nécessite la génération de clés ssh entre le coordinateur et les trois sites de calcul. Par ailleurs, XtremWeb offre une interface web qui est la partie publique de la plateforme. Ces pages peuvent être utilisées pour des demandes de participation au système, le téléchargement d'archive .jar du client ou worker par d'éventuels participants, et consultation de quelques statistiques.

Nombre	CPU	Mémoire
45	AMD Athlon(tm), 2.8GHz	1GB
32	AMD Athlon(tm), 1.8GHz (7 bi-processors)	1GB
13	AMD Athlon(tm), 2.2GHz	1GB
12	AMD K7, 600MHz	256MB
10	Pentium 4, 2GHz	256MB
12	Pentium III (Katmai), 500MHz	128MB
9	Pentium II (Deschutes), 400MHz	128MB

TAB. 4.2 – Configuration des machines à Paris-XI

Nombre	CPU	Mémoire
16	Bi-processor Xeon, 2.4GHz	1GB
10	Bi-processor AMD Athlon, 1.8GHz	1.5GB
8	Quadri-processor Pentium II (Deschutes), 450MHz	2GB

TAB. 4.3 – Configuration des Clusters à Tsukuba

Les plateformes d'expérimentations sont composées d'un coordinateur central et de noeuds de calcul sur les trois sites d'exécution. Tous les noeuds de calcul sont sous Linux. Les archives Java pour workers sont copiés (via scp) sur les sites distants. Le coordinateur est placé à l'École Polytechnique Universitaire de Lille sur une machine serveur dont les caractéristiques ont évolué durant ces deux dernières années : Intel Pentium 4, 2.4GHz et 512M de mémoire vive, puis Intel Xeon, 2.66GHz et 1G de mémoire vive. Les tableaux 4.1, 4.2 et 4.3 donnent la description des ressources sur les trois sites. À Lille, un ensemble de 128 PCs non-dédiés, principalement des machines d'enseignement à l'École Polytechnique Universitaire de Lille. À l'université de Paris-XI, un ensemble de 133 ressources hétérogènes non-dédiées, et à l'université

de Tsukuba, un ensemble de trois Clusters ; Dennis (16 noeuds bi-processeurs), Alice (10 noeuds bi-processeurs) et Cosmo (8 noeuds quadri-processeurs).

La portabilité des applications matricielles de base est assurée par des éditions de lien statique des codes car toutes les machines ne contiennent pas les BLAS et LAPACK. Ces binaires sont stockés dans la base de données du coordinateur et fournis lors de la première demande du worker. Par ailleurs, XtremWeb ne met pas ces binaires à jour sur les workers en cas de changement sur le coordinateur. Il a fallu écrire un script de "nettoyage" (similaire aux précédents) en cas de mise à jour de binaires sur le coordinateur avant de relancer une application (faute de quoi, d'anciens binaires seront utilisés).

4.4 Collecte d'informations

La collecte d'informations sur les plateformes de calcul a été faite par deux outils, NWS (Network Weather Service) et Netperf. L'installation et l'administration de ces outils a été réalisé dans le but de scruter le comportement du réseau et des machines de la plateforme de calcul en vue de mesurer la latence et bande passante utilisés par nos applications, ainsi que la charge des unités centrales de traitement et espace mémoire (ou disque) disponible sur les neouds.

NWS [WSH99][Wol][use] est un outil de mesure et prévision des caractéristiques dynamiques d'un ensemble de machines en réseau. Ce système répartit des senseurs et prédicteurs statistiques et permet de centraliser l'état présent de la plateforme et de fournir des prévisions au sujet des évolutions futures en appliquant un ensemble de traitements statistiques sur les mesures passées. Même en l'absence de demandes, le système continue donc à faire des mesures régulières. Cet outil est essentiellement prévu pour l'ordonnancement et est utilisé à cet effet par AppLeS, Globus et Netsolve notamment.

Un système NWS est basé sur 4 processus ; un serveur de nom qui est l'annuaire du système et qui permet de connaître les différents processus disponibles sur un hôte donné, un ou plusieurs processus d'état ou mémoire qui fournissent le stockage des données et mesures rassemblées par le déploiement de NWS, des senseurs qui fournissent ces mesures, et des prédicteurs qui produisent des prévisions dynamiques basées sur des mesures historiques. NWS a été utilisé selon le schéma de la figure 4.4. Le déploiement des différents processus n'est pas automatique et c'est une tâche relativement longue et fastidieuse. Le serveur de nom (`nws_nameserver`) tourne sur le coordinateur d'XtremWeb. Il contient aussi un processus `nws_memory` pour le stockage et un processus `nws_sensor` de collecte de mesures. Sur les deux sites distants, chaque machine contient un senseur et une machine au moins un processus d'état qui collecte les mesures des autres noeuds du site. Les noeuds à Lille contiennent des senseurs.

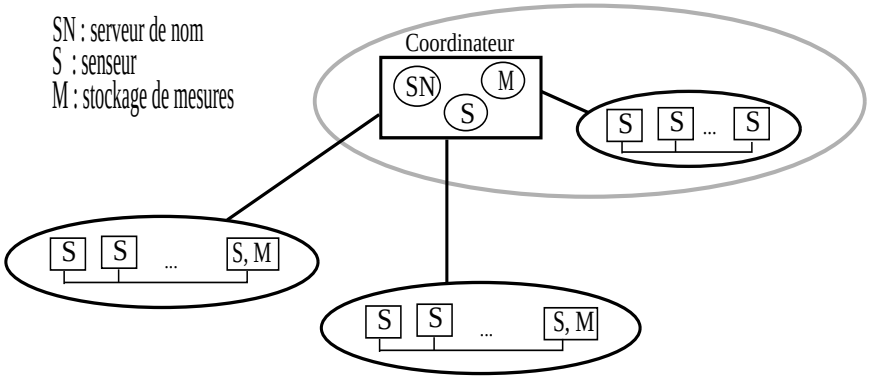


FIG. 4.4 – Schéma d'utilisation de NWS.

Vu le caractère centralisé d'XtremWeb, les mesures de latence et bande passante entre noeuds du système ne sont d'aucune utilité. De même pour la prédiction de performances car aucune interactivité n'est possible avec le lancement d'applications sous XtremWeb, qui au reste, ne gère absolument pas la localité des tâches ou des données. Les seules mesures significatives sont donc les débits et latences entre le coordinateur et les noeuds sur les différents sites. L'outils Netperf[2] est moins complexe à employer pour la réalisation de ces mesures.

Netperf [net] est un outil de mesure de performances réseau plus facile à déployer et à utiliser que NWS mais avec moins de fonctionnalités. Il permet de mesurer les flux TCP et UDP entre deux entités dans le réseau. Il comporte aussi des tests optionnels de mesure de performances pour DLPI, Unix Domain Sockets, l'API ATM de Fore systems, ou Hi Performance Parallel Interface de HP. Pour utiliser Netperf, il est nécessaire de lancer le programme serveur `netserver` (comme service `inetd` ou en ligne de commande). Les mesures s'effectuent par le programme `netperf` qui établit une connexion de contrôle TCP au système distant (basée sur les sockets BSD), indépendamment du type de test à exécuter. Une fois cette connexion en place et les informations de configuration passées, une connexion séparée est ouverte pour les mesures, utilisant les APIs et protocoles appropriés pour le test demandé.

4.5 XtremWeb-CH

XtremWeb-CH[3] est une version décentralisé d'XtremWeb. Cette version dérivée est développée au HES-SO à l'université de Genève. Elle consiste en un module d'ordonnancement au-dessus d'XtremWeb qui insère les tâches à la volée à travers une description XML, figure 4.5. XtremWeb-CH lance les tâches sans dépendance,

[2] http ://www.netperf.org/netperf/NetperfPage.html
[3] http ://www.xtremwebch.net/

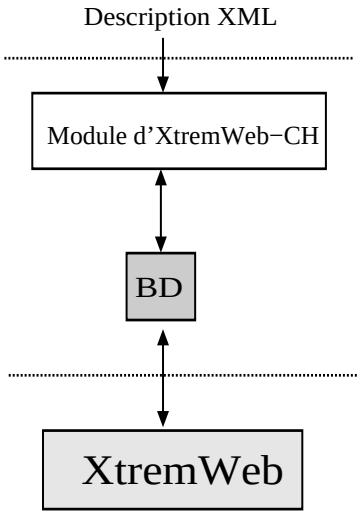


FIG. 4.5 – Organisation d'XtremWeb-CH.

récupère les résultats et met à jour les tables de la base de donnée sur le coordinateur. Dans cette version les workers peuvent communiquer directement, seules des références sur les données sont envoyées au coordinateur.

L'originalité de cette version, mis à part le fait qu'elle soit décentralisé, réside dans l'utilisation d'une description XML. Cette description prend en compte les dépendances et synchronisations entre tâches et la réutilisation de résultats intermédiaires, alors que ceci est fait 'à la main' dans la version centralisée d'XtremWeb. Cette façon de décrire les applications nous semble plus pratique que l'interface Java de la version centralisée. Toutefois, cela peut générer un surcoût important par rapport à la version centralisé lié à ce module, et notamment au traitement additionnel sur la base de donnée [AP]. Cette version a été installée et évaluée sur la plateforme de calcul local de 128 machines.

4.6 Grid'5000

Grid'5000[4] est une plateforme matérielle et logicielle, interconnectant huit Clusters de PCs de grande taille (entre 100 et 1000) géographiquement réparties en France. Chaque Cluster de la plateforme est administré localement mais une politique globale de fonctionnement est assurée par des structures nationales ; administrative et scientifique. Les noeuds de Grid'5000 sont connectés au réseau Renater (par VLANs utilisant MultiProtocol Label Switching au niveau 2). L'objectif principal de ce projet est de doter la communauté de recherche en Grid d'une plateforme

[4]http ://www.grid5000.org/

expérimentale configurable. Un utilisateur pourra utiliser un ensemble d'outils logiciel de réservation, de déploiement et de contrôle pour configurer une partie de la plateforme avec son OS (noyau et/ou distribution), ses protocoles réseau, ses middlewares et environnements d'exécution, pour la conduite de ses expériences. La plateforme devrait donc permettre de reproduire les conditions expérimentales et les résultats des tests (en particulier intra-site), et fournir des mécanismes de mesure, stockage et visualisation. Typiquement, les expériences visées incluent la conception et évaluation de protocoles, de middlewares, de techniques de tolérance aux fautes, conception algorithmique pour le calcul haute performance, évaluation de performances pour le calcul scientifique, etc.

Au niveau application et exploitation algorithmique pour le calcul scientifique, les problématiques relevant du calcul sur la grille sont bien connues. Il s'agit essentiellement d'adapter les modèles de calcul (et donc les middlewares) à la dynamicité et l'hétérogénéité à tous les niveaux (du réseau à la hiérarchie mémoire des machines), et aussi de permettre le passage à l'échelle. Les expérimentations algorithmiques en vraie grandeur sur un outil configurable, flexible et reproductible sont donc essentielles. L'étude des techniques de distribution (et probablement re-distribution) inhomogène des données pour le calcul matriciel sous des contraintes d'optimisation des communications, de temps de calcul ou d'équilibrage de charge, en est un exemple (qui est par ailleurs un problème combinatoire très difficile). Nous utiliserons dans cette thèse des noeuds de Grid'5000 (répartis sur 4 sites, notamment Grid eXplorer) pour des séries de tests.

4.6.1 Grid eXplorer

Grid eXplorer[5] est le noeud de Grid'5000 situé à l'IDRIS, sur le campus d'Orsay. Ce cluster, acquis dans le cadre de l'ACI Masse de Données, possèdera le plus grand nombre de CPUs (1088). Étant donné l'homogénéité matérielle des noeuds (bi-processeurs AMD opteron 2GHz, et 2Go de mémoire interconnecté par une infrastructure réseau Gigabit Ethernet), Grid eXplorer est plus un outil d'émulation qu'une grille expérimentale réelle.

Grid eXplorer a été le point d'accés à Grid'5000 (le noeud de Lille n'étant pas encore opérationnel). Les tests sur cet outils se déroulent généralement en 3 étapes : 1) la réservation des noeuds de calcul déployables nécessaires, en utilisant l'outil `OAR` 2) la reconfiguration des noeuds pour les conditions de l'expérience. Cela inclut l'installation de logiciels spécifiques requis pour l'application, l'enregistrement de l'image associée et le déploiement en utilisant l'ensemble des commandes de `kadeploy`, 3) et finalement le lancement des applications réelles sur les noeuds déployés et tests de performances. La deuxième étape consiste à augmenter une distribution debian minimale sur les noeuds de calcul avec les outils nécessaires à nos tests, en l'occurrence

[5] http ://www.lri.fr/~fci/GdX/

l'environnement XtremWeb et tous les softwares sous-jacents requis (compilateurs gnu, mysql, une jdk...).

La difficulté majeure en ce qui concerne XtremWeb est le besoin d'un point d'attache fixe qu'est le coordinateur. Ceci pose problème étant donné qu'on est pas sûr qu'un noeud donné soit systématiquement libre et allouable au moment des tests. Pour résoudre cela, le lancement des workers se fait via une archive java qui est modifiée avec un nouveau fichier de configuration comprenant le nouveau coordinateur, et ré-enregistrement de l'environnement avant chaque déploiement. Cet environnement est 'allégé' et contient juste une jdk et l'archive java du worker. Le lancement d'XtremWeb sur les noeuds déployés posait aussi problème car l'outil `kadeploy` modifie les clés `ssh` autorisées pour l'utilisateur root afin de permettre de redémarrer les noeuds à la fin de la réservation sans faire systématiquement un redémarrage matériel. Ces clés devaient être utilisées en vue de lancer des scripts de déploiement d'XtremWeb à partir des machines frontales. Ceci a été résolu en utilisant un login différent (celui de l'utilisateur xtremweb) en lui attribuant les droits nécessaires via `sudo`. Une autre *postinstall* de `kadeploy` qui garderait ces clés pourrait aussi être utilisée. Les étapes citées auparavant (réservation, l'installation de packages nécessaire et enregistrement de l'environnement) sont à refaire sur chaque site de Grid'5000 participant au calcul.

4.7 Première expertise

Le déploiement et test des outils de grille et plateformes expérimentaux utilisés dans cette étude impliquent un travail de mise au point et mise à jour continues qui posent par ailleurs des problèmes système et logiciel fréquents (incompatibilité entre versions par exemple notamment concernant l'API client, différence de configuration entre sites et nécessité d'adaptation pour les déploiement d'environnements...). Néanmoins, ces outils sont nouveaux et expérimentaux et malgré le soin certain qui est apporté à leurs réalisation et mise en place, le difficile travail de débogage demeure important pour qu'ils soient conformes au cadre final recherché et implémentés dans des versions stables et durables[6]. Par ailleurs, la description et expertise des outils utilisés nous permet d'ores et déjà de mettre en évidence certaines caractéristiques liées à la conception et outils sous-jacents tel que le langage de programmation par exemple.

Pour XtremWeb, le langage Java a été certainement choisi dans un souci de portabilité, pour ces mécanismes de sécurité et de gestion des exceptions, il est aussi orienté objet et donc facilement réutilisable, et offre une relative facilité de programmation (par rapport à C++ par exemple), et n'utilise plus de pointeurs mais des

[6]Plus de vingt versions d'XtremWeb depuis deux ans dont plus d'une dizaine utilisées et qui nécessitent à chaque installation un travail de mise à jour plus ou moins important.

références. Ce langage est aussi très orienté réseau et fournit un bon support d'invocation à distance (RMI). Néanmoins, aux quelques avantages énumérés ci-dessus il faut ajouter les limites qui peuvent affecter les performances des applications matricielles. En premier lieu, les temps d'interprétation indissociable à ce langage malgré l'introduction d'améliorations à chaque sortie d'une nouvelle version de Java à tous les niveaux du système d'exécution, tel que des mécanismes comme la compilation à la volée (*Just-in-Time*). Aussi, le Garbage Collector (gestion automatique de la mémoire) citée généralement comme un avantage et qui n'est absolument pas maîtrisable et peut décider de son propre chef de s'exécuter à un moment inopportun ou au contraire ne pas le faire au moment opportun, car même si les tâches matricielles de base sont écrites en langage C (utilisant les BLAS et LAPACK), les applications matricielles sont écrites en Java et la création de grand nombre d'objets ou de grandes tailles peut conduire à une consommation mémoire excessive qui dégrade considérablement les performances à cause de la pagination mémoire (il n'est donc pas possible d'éviter cela en gérant explicitement la memoire du client), et dans le pire des cas, cela peut engendrer l'arrêt total du programme. Aussi, les vérifications supplémentaires lors de l'exécution (comme le test de dépassement de capacité d'un tableau) induisent des overheads additionnels qui affectent forcement les performances.

Par ailleurs, l'organisation d'XtremWeb-CH (la version décentralisé d'XtremWeb décrite plus haut) introduit des étages supplémentaires de description via son module d'ordonnancement, le schéma XML et la gestion de références sur les données. Il est évident que ceci génère des overheads qui doivent être absorbés par des tâches de calcul plus importantes et avec peu ou pas de communications, ce qui n'est pas forcement le cas des applications de l'algèbre matricielle. D'un autre côté, les modèles mathématiques et de simulation pour les grilles évoqués ici se basent souvent sur des hypothèses assez restrictives et leurs degrés de fiabilité (ou validation) est difficile à estimer notamment en regard de la réalité dynamique et fortement hétérogène du Grid. Les hypothèses restrictives et modèles simples utilisés sont difficilement vérifiable en pratique. Ceci est en l'occurrence le cas de tous les systèmes décrits plus haut. En outre, l'impact sur les applications réelles de ces hypothèses reste à étudier.

4.8 Conclusion

Nous avons décrit dans ce chapitre nos plateformes d'expérimentation ainsi que les middlewares et outils d'expérimentation et d'administration utilisés. Nous avons également rappelé que l'étude et la recherche en Grid se base sur plusieurs outils : les modèles mathématiques, la simulation/émulation, et le déploiement sur grilles expérimentales, qu'il convient de séparer des grilles de production, qui sont des grilles applicatives. En somme, dans un schéma classique, les trois premières étapes préludent à la réalisation d'outils et déploiement en production.

L'outil XtremWeb décrit ici présente des caractéristiques intéressantes pour le calcul scientifique, notamment la compression des données, l'exécution d'applications natives (et donc la possibilité d'utiliser des bibliothèques scientifiques et codes matriciels matures existants), et des mécanismes de protection des ressources et sécurisation des communications. Néanmoins, cet outil est expérimental et les restrictions et contraintes liées à son organisation actuelle et outils sous-jacents, ainsi que l'absence de politique d'ordonnacement limitent grandement son champ d'application ; typiquement adapté à un parallélisme massif, multi-paramètres ou maître-esclave ne nécessitant pas de communications. En outre, nous avons établi une analyse relative aux outils expérimentaux utilisés qui sont relativement nouveaux et qui font toujours l'objet de développement, extensions et révisions très actifs. Dans le chapitre suivant, nous commençons par une approche classique de test sur les plateformes de calcul décrites ici et sur Grid'5000. Nous proposerons par la suite un paradigme de calcul mieux adapté à l'algèbre matricielle sur grilles de calcul non-dédiées.

Chapitre 5

Algorithmique numérique sur la grille : une approche classique

5.1 Introduction

Si de gros efforts sont actuellement effectués au niveau matériel et middleware (intergiciel) afin de permettre une bonne utilisation des environnements Grid, l'écriture et optimisation des applications sur de telles architectures restent encore très délicates, principalement en raison de l'hétérogénéité réseau et des capacités de calcul et stockage des différentes machines. Les performances des communications et du calcul ne sont pas aussi stables et fiables que celle d'un supercalculateur ou un cluster de PCs. D'ailleurs le but n'est pas d'atteindre les performances de tels systèmes, mais d'avoir la meilleure exploitation possible des grilles informatiques en proposant notamment de nouvelles approches et modèles de calcul bien adaptés, et des langages et outils d'aide pour l'utilisateur final.

Ceci dépend aussi de l'application visée et il est nécessaire de concevoir des algorithmes adaptés, ou adapter les algorithmes existants, à ces environnements. Néanmoins, vu le caractère extrêmement hétérogène et dynamique des grilles, il serait possible de proposer plusieurs schémas de calcul/distribution, adaptés à chacun des problèmes rencontrés (équilibrage de charge, ordonnancement, communications...), pour une même application. Obtenir un bon compromis pour de bonnes performances qui soit assez durable et générique est donc un défi très difficile.

Dans ce chapitre, nous présentons des implémentations sur des grilles de calcul de deux applications d'algèbre linéaire, le produit matrice-vecteur et l'inversion de matrice par Gauss-Jordan, en utilisant des distributions par blocs classiques. Nous introduisons ensuite la notion de stockage persistant, incluant l'anticipation de migration et le clouage des données (section 3.3.3), et présentons une validation expérimentale de l'efficacité de ces schémas de distribution de données.

5.2 Calcul matriciel sur la grille

L'évolution des environnements parallèles a motivé la conception et l'extention des techniques et paradigmes de programmation pour permettre de prendre en compte les caractéristiques inhérentes à chacun d'entre eux. Les techniques d'ordonnancement et d'équilibrage de charge par exemple ont été largement étudiées et sont relativement maîtrisés dans le cas homogène, pour supercalculateurs ou clusters, mais sachant que les plateformes de calcul du futur pourraient être fortement hétérogène, ces schémas ne sont plus adaptés. Néanmoins, pour nos application, nous avons choisi de suivre une distribution homogène et statique pour les étapes de calcul et d'y introduire des optimisations de placement des données, sans utiliser des techniques de re-distribution, de recouvrement ou de découpage hétérogène.

Le développement de logiciels numériques pour environnements parallèles classiques a fait l'objet de recherches abondantes depuis quelques dizaines d'années [DS86][Pet88][DW95][DD00]...Seulement, les hypothèses relatives aux architectures servant de base à ces implémentations ne sont plus réalistes et les problèmes liés à l'algorithmique sur la grille sont encore loin d'être maîtrisés. Néanmoins, plusieurs schémas et techniques d'optimisation en environnements hétérogènes ont été proposés en algèbre linéaire, notamment pour le produit matriciel ou la factorisation LU [Leg03][Car00][DHW97]. Ces environnements étaient cependant très restreints (quelques dizaines de machines au plus) et ne sont pas comparables à une grille. Aussi, les solutions proposées ne sont efficaces que sous certaines conditions, pour des cas simple d'hétérogénéité et pour des applications données. Il n'existe pas de solution optimale générique. Dans nos implémentations, à défaut de pouvoir prendre en compte tous les aspects de l'hétérogénéité de la grille, nous proposons, dans un premier temps, une approche tout à fait classique.

Le middleware utilisé, décrit dans le chapitre précédent, présente un schéma de calcul simple de type multi-paramètres et maître-esclave, est peu évolué et n'offre pas d'ordonnancement adapté, ni de gestion de la localité des données, qui sont deux paramètres très importants pour le calcul matriciel. Ce type d'applications nécessite généralement plusieurs phases de communications entre coordinateur et workers et n'est donc pas nécessairement adapté à ce type de paradigmes de calcul, notamment en raison du caractère centralisé des communications et de l'hétérogénéité de la plateforme (temps de communication non uniformes, différence de puissance et capacités entre noeuds...). Ceci motive l'introduction d'optimisations de placement des données et de gestion des contraintes mémoire dans cette étude. Dans la suite de ce chapitre, nous présentons nos implémentations classiques et résultats des tests, ainsi que l'apport d'un placement efficace des données sur les performances des applications sur la grille.

5.2.1 Le produit matrice-vecteur par blocs

Le produit matrice-vecteur $Ax = b$ est un noyau de l'algèbre linéaire, et notamment des méthodes itératives. Historiquement, le produit matrice-vecteur est souvent une des opérations primaires autour de laquelle tous les supercalculateurs sont évalués [Don87]. Dans le contexte du calcul scientifique, les algorithmes itératifs sont bien adaptés à une grande classe de problèmes (résolution de systèmes linéaires, problème de valeurs propres...). Pour ces méthodes itératives, la matrice A reste inchangé, ceci permettra le clouage des blocs de données et l'anticipation des migrations. Cela veut dire qu'à partir de la deuxième itération, les blocs de la matrice seront présents là où ils seront utilisés et le produit matrice-vecteur nécessitera beaucoup moins de communications. Nous évalurons les performances des produits correspondants.

Nous décrivons deux implémentations basées sur deux distributions de blocs ; bidimensionnelle et unidimensionnelle. Les figures 5.1 et 5.2 montrent le découpage en blocs des matrices de test pour les deux versions. La première version présente des dépendances (figure 5.3) et nécessite deux phases de communications alors que la version 2 ne présente qu'une seule phase de communications, initiales (envoie des blocs de lignes et du vecteur) et finales (récupération de blocs du vecteur résultat). La première découpe (blocs carrés) est plus classique et nombre d'applications matricielles par blocs utilisent et fournissent des matrices sous cette forme. Les produits se déroulent de la façon suivante.

On dispose au minimum de p^2 noeuds sur la plateforme de calcul. Soit A (respectivement x) une matrice carré (respectivement vecteur) de dimension N. Le premier algorithme se déroule en deux étapes ; dans la première $p \times p$ blocs de matrice de dimension $n_1 - par - n_1$ ($n_1 = \frac{N}{p}$) et p blocs de vecteur sont distribués sur p^2 noeuds. Le coordinateur distribue donc les blocs correspondants de A et x pour la multiplication (le binaire de multiplication envoyé par le coordinateur utilise la routine `DGEMV` de BLAS2 [bla]). Dans la deuxième étape ; les blocs vecteurs résultats ayant le même indice de ligne sont distribués pour la somme. Cette étape dépend donc des produits effectués sur une même ligne et peut être effectuée indépendamment des autres. Des communications finales sont nécessaires pour récupérer les blocs du vecteur résultat.

Le deuxième algorithme se déroule en une seule étape ; p^2 blocs de A de dimension $n_2 - par - N$ ($n_2 = \frac{N}{p^2}$) et le vecteur x sont distribués sur p^2 machines. Le coordinateur d'XtremWeb envoie les blocs de lignes de A et le vecteur x sur la plateforme de calcul pour la multiplication (utilisant BLAS2), et les résultats retournés sont les blocs du vecteur résultat (de taille n_2).

Dans ces implémentations, le coordinateur est aussi client. Il gère les dépendances et synchronisations, et achemine toutes les communications. L'API Java permet de spécifier un environnement d'exécution, d'envoyer des tâches et de récupérer les

résultats. Par contre, les dépendances et synchronisations, et la réutilisation des fichiers résultats sont beaucoup plus délicates à gérer et ne sont pas prévus dans cette API. Ceci est géré via des tests sur la fin des tâches, une gestion explicite des fichiers zippés (pour récupérer les résultats intermédiaires), et la mise en place de délais. À noter aussi qu'une gestion explicite des fautes et re-soumission des tâches s'est avérée nécessaire dans ces implémentations à cause des échecs des programmes clients causés par l'arrêt de certaines tâches de l'application sur des noeuds défaillants. XtremWeb ne semble pas supporter ces échecs et ceci cause l'arrêt de l'application cliente. Il fallait donc supprimer les tâches en question de la base de données, les ré-initialiser et les re-soumettre dans le corps du code client Java.

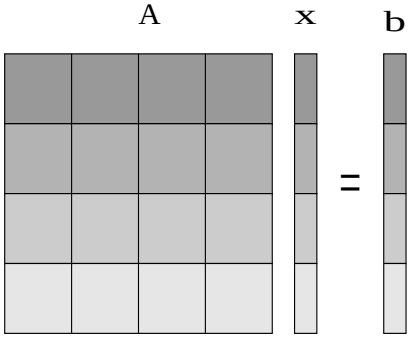


FIG. 5.1 – Découpage bidimensionnel (version 1).

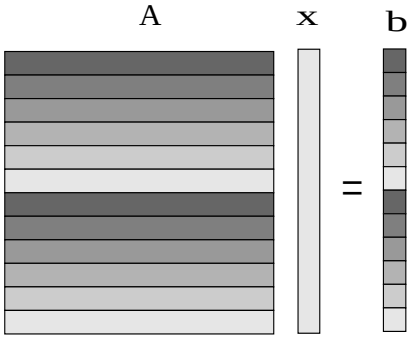


FIG. 5.2 – Découpage unidimensionnel (version 2).

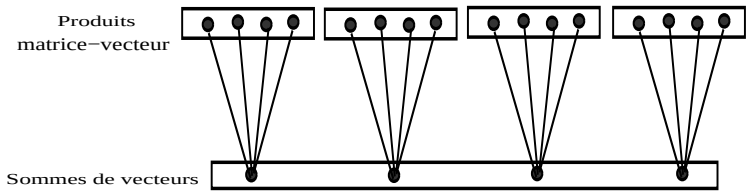


FIG. 5.3 – Graphe de tâches pour $p = 4$ (version 1).

5.2.1.1 Résultats et analyse

Ces deux algorithmes ont été déployés en utilisant deux configurations de la plateforme de calcul (décrite dans le chapitre précédent). La première série d'expériences a été réalisée sur un réseau local (les 128 noeuds de calcul non-dédiés à Lille). Le coordinateur est une machine dédié situé à l'École Polytechnique Universitaire de Lille. La seconde série d'expériences a été réalisée sur un réseau étendu (*wide-area*) composé des sites de Lille et de l'Université de Paris-XI à Orsay. La description des noeuds de calcul est donnée en section 4.3. La bande passante calculée entre le coordinateur et les noeuds à Lille est 9.4Mb/s (avec une latence moyenne de 0.6ms) et 6.2Mb/s entre le coordinateur et les noeuds à Orsay (avec une latence moyenne de 5ms). La mesure des temps de calcul dans les programmes clients Java utilise la méthode statique et native `currentTimeMillis()` de la classe `System`.

Les matrices de test sont denses et de taille $N = 30000$ ($\sim$ 7Go). Comme nous l'avons fait remarquer précédemment, la distribution est homogène et chaque noeud reçoit la même quantité de données. Différentes tailles de bloc ont été utilisées, pour la première version, $n_1 = 1500$, 3000 et 3750. Pour la deuxième version $n_2 = 300$ (taille de bloc $300 - par - 30000$). Les matrices sont générées aléatoirement (flottants en double précision) et découpées préalablement en blocs sur le coordinateur avant exécution. À chaque exécution, on introduit entre 72 et 420 tâches de calcul (multiplication matrice-vecteur BLAS2, ou somme de blocs de vecteur) dans le réseau de calcul sous XtremWeb. Ce nombre de tâche correspond à p^2 multiplications matrice-vecteur, plus p sommes de p blocs de vecteur pour la version 1. Pour la version 2, le nombre de tâche correspond à p^2 multiplications matrice-vecteur.

Deux dispositions des données ont été considérées. D'abord, toutes les communications et transferts des blocs sont effectués, ensuite on simule le stockage persistant des blocs de la matrice A en les générant sur les noeuds de la plateforme de calcul étant donné que le middleware ne gère pas la localité. En effet, comme discuté plus haut dans ce chapitre, la matrice A reste inchangé dans les méthodes itératives, cela permet d'exécuter les produits consécutifs, sans ré-envoie des blocs, sur les noeuds

où ils se trouvent préalablement (seuls les blocs de vecteurs actualisés sont réenvoyés).

Les figures 5.4 et 5.5, et le tableau 5.1 montrent les moyennes et valeurs minimums des temps d'exécution (incluant calcul et communication), sur les plateformes de calcul locale et étendue. La meilleure moyenne pour le produit de taille 30000 a été obtenue avec une taille de bloc de 3750 en utilisant le stockage persistant des données. La meilleure moyenne sans stockage persistant, c'est-à-dire avec gestion de toutes les communications, est obtenu avec une taille de bloc de 3000. Ceci démontre l'importance du choix de la taille des données, notamment par rapport à la disposition et la gestion de la localité, mais aussi par rapport à l'*overhead* du système qu'il doit recouvrir. En effet, on remarque l'augmentation du temps de calcul entre les tailles 3000 et 3750 dans le cas de gestion de tous les transferts, alors que cela diminue en utilisant le pré-déploiement des blocs. Ceci est indispensable, car il serait extrêmement pénalisant d'envoyer les blocs des données à chaque itération dans une méthode itérative. En l'occurrence, le choix se porterait sur une taille de bloc de 3750 pour l'exécution d'une méthode itérative sur notre plateforme (dans le cas où le middleware gère la localité des données avec un coût minimum de gestion des références). Par ailleurs, le produit avec taille de bloc 1500 a de très mauvaises performances car le nombre de noeuds nécessaire (ou minimum) pour espérer avoir de meilleurs temps de calcul est de 400.

Tailles des blocs	Avec gestion de toutes les communications		En utilisant le stockage persistant		Nombre de tâches matricielles
	moyenne	minimum	moyenne	minimum	
Sur le réseau local					
1500	1072 s	813 s	663 s	337 s	420
3000	239 s	174 s	110 s	87 s	110
3750	726 s	604 s	94 s	74 s	72
Sur le réseau étendu (*wide-area*)					
3000	436 s	355 s	283 s	196 s	110
3750	995 s	827 s	201 s	136 s	72

TAB. 5.1 – Temps d'exécution et nombre de tâches matricielles. Plateformes de calcul sur le réseau local à Lille, et étendu ; Lille et Orsay.

Dans la version 2, il n'y a pas de communications entre tâches. À chaque test, 100 tâches de calcul de produit matrice-vecteur sont exécutées. La moyenne de calcul sur la plateforme locale est de 240 secondes (avec un temps minimum de 153 secondes), et 526 secondes sur la plateforme WAN (avec un temps minimum de 393 secondes). Ces résultats sont sensiblements analogues à la version 1, pour le même

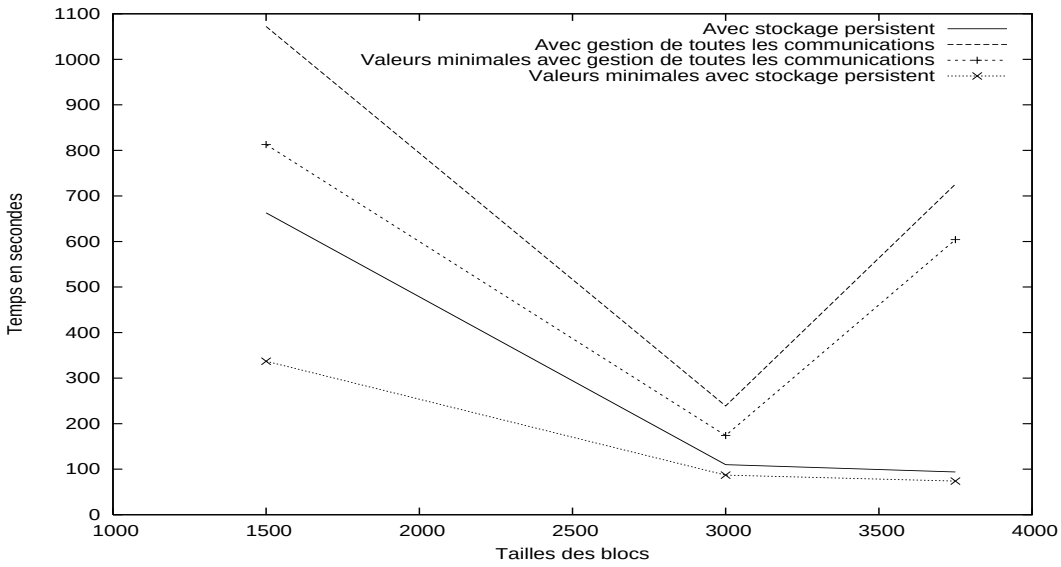


FIG. 5.4 – Produit matrice-vecteur de taille 30000 sur la grille locale (128 noeuds).

nombre de flottants par bloc (taille de bloc carré $3000-par-3000$). L'interprétation de ces résultats est difficile car cette version devait être meilleure, notamment en raison des dépendances moindres dans le code (mais avec plus de données envoyées par tâche de calcul). Ceci montre que le paradigme classique maître-esclave n'est pas nécessairement adapté et qu'il est nécessaire d'optimiser les besoins en communications.

En effet, le rapport calcul/communication par tâche de calcul est très petit, il avoisine 2 pour les deux versions, tableau 5.2. En utilisant le stockage persistant des blocs de la matrice, ce rapport devient égal à $\frac{N}{p}$ pour la première version et $\frac{2N}{p^2}$ pour la version 2. On aurait donc plutôt intérêt à diminuer la taille des blocs (p) dans la mesure où la taille de la plateforme le permet (i.e. p^2 noeuds de calcul au minimum), car sinon nous avons constaté que la taille de bloc de 1500 donne de moins bons résultats par rapport aux blocs de tailles 3000 et 3750. Reste que le stockage persistant des blocs de données donne de meilleurs temps d'exécution par un facteur de 1.5 à 7.7 par rapport à la version gérant toutes les communications. Le placement efficace des données est donc d'importance primordiale dans le calcul matriciel sur la grille.

Nous avons testé la version 1 du produit de taille 30000 (avec $n_1 = 3000$) sur la version décentralisée XtremWeb-CH en utilisant les 128 noeuds de la plateforme

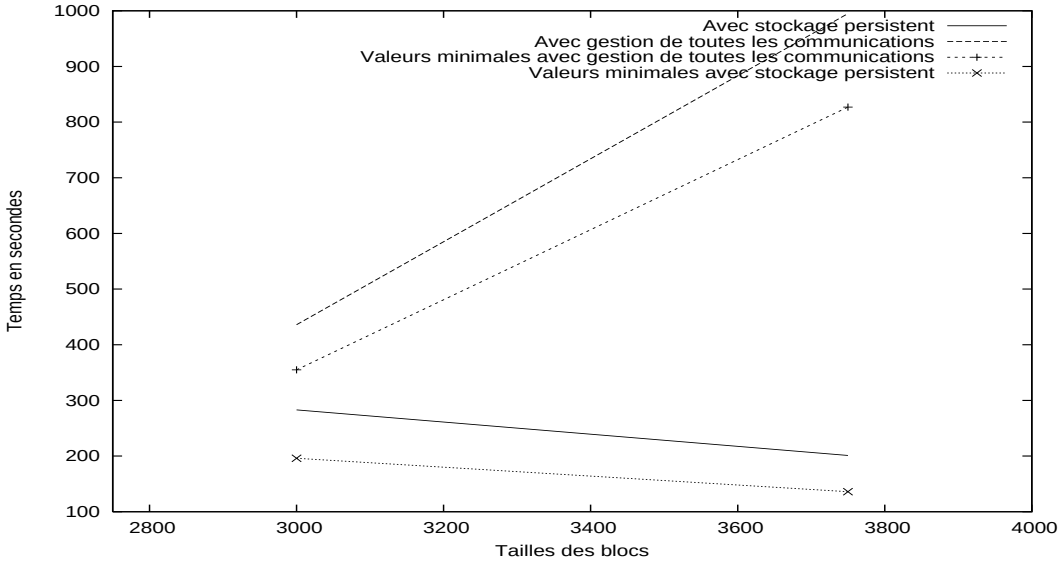


FIG. 5.5 – Produit matrice-vecteur de taille 30000 sur la grille étendue (261 noeuds).

	Version1	Version2
Sans stockage persistant	$\frac{2pN^2-N}{pN^2+2p^2N}$	$\frac{2N^2-N}{N^2+p^2N}$
Avec stockage persistant	$\frac{N}{p}$	$\frac{2N}{p^2}$

TAB. 5.2 – Rapports calcul/communication pour chaque tâche de calcul.

locale. Le produit matrice-vecteur est décrit en XML, et contrairement à l'API Java d'XtremWeb, les dépendances et synchronisations peuvent être exprimées naturellement. La syntaxe XML apporte plus d'aisance et une meilleure lisibilité pour le développement d'applications et description de graphes de tâches. Dans cette version, les workers peuvent communiquer directement, seules des références sur les données sont envoyées au coordinateur. Cependant, cette version a donné de très mauvaises performances par rapport au même produit sous la version classique XtremWeb, qui est en moyenne meilleure avec un facteur de 2.7 (239 contre 642 secondes). Ceci laisse penser que les traitements additionnels générés par le module d'ordonnancement d'XtremWeb-CH notamment sur la base de données, et l'étage supplémentaire introduit par la description XML, sont très pénalisants. L'*overhead* associé est donc important. La taille des tâches de calcul est néanmoins un facteur déterminant et peut influer sur la différence de performances.

Nous avons finalement testé le produit matrice-vecteur en utilisant un système

d'appels RPC ; OmniRPC, sur le réseau local à Lille. Cet environnement peut gérer la persistance des données en associant un *handle* aux tâches de calcul. Il est donc possible de placer les tâches selon les données présentes sur les noeuds de calcul marqués auparavant. Les temps de calcul pour la version 1 de taille $N = 30000$ avec gestion des *handles* est de 210 secondes pour des blocs de taille 3000, et 231 secondes pour des blocs de taille 3750. Ces temps de calcul sont jusqu'à 20 fois moins importants sans la gestion de *handles* (respectivement 10.5 secondes et 19.4 secondes). Ceci montre que l'*overhead* associé à la gestion des références est très important car ces versions n'envoient pas de données (génèrent les blocs sur les noeuds de calcul). Par ailleurs, ce système n'est pas tolérant aux fautes et s'est avéré limité et peu fiable pour l'envoie de données volumineuses. La taille maximale qu'on a pu atteindre pour le produit avec envoi est de $N = 10000$ et le temps de calcul est 771 secondes (version 2, blocs de taille $1000 - par - 10000$). Nous remarquons donc que l'utilisation des *handles* et le transfert d'importantes quantités de données pénalisent ce système et augmentent considérablement les temps de calcul.

Les résultats de cette section montrent que XtremWeb fournit de meilleurs temps de calcul que XtremWeb-CH ou OmniRPC. Ceci est dû à des overheads plus importants sur ces deux derniers environnements liés notamment à la gestion des références et aux niveaux supplémentaires de traitement sur la version décentralisée. Néanmoins, la comparaison avec OmniRPC est faite seulement sur des exécutions sans envois de données à cause des limites de scalabilité de cet environnement. D'autres limites et contraintes pour le passage à l'échelle existent aussi dans XtremWeb et seront décrits dans la suite de ce chapitre.

5.2.2 L'inversion de matrices par Gauss-Jordan par blocs

La deuxième application test est l'inversion de matrice. Les algorithmes permettant de résoudre ce problème sont nombreux. Nous avons choisi l'une des méthodes directes les plus classiques ; Gauss-Jordan par bloc, pour sa stabilité et sa complexité moindre [Pet88][MTP00].

Soit la matrice réelle A d'ordre N. A est dense et générale. Il s'agit donc de trouver, quand elle existe, la matrice B telle que $B \times A = A \times B = I$, où I est la matrice identité d'ordre N. La matrice A est découpée en $p \times p$ blocs de dimension n et la matrice inverse B est progressivement construite par blocs ($p \times p$ de taille n). Notre réalisation se fait donc sans considérer des matrices temporaires et est donc mieux adaptée à la grille. Les parties de l'algorithme comprises entre deux inversions de blocs diagonaux sont appelées des étapes. L'algorithme de base par bloc est composé de p étapes et se déduit de celui par point en remplaçant les opérations scalaire par des opérations sur les blocs. L'algorithme utilisé est donné ci-dessous.

L'originalité de cet algorithme réside dans le fait qu'il construit la matrice cible

progressivement sans utiliser de blocs temporaires. Chaque étape k (de 0 à $p-1$) de l'algorithme est composée de 3 parties. La première est l'inversion du bloc pivot A_{kk}. La deuxième partie est composée de $2(p-1)$ produits de blocs ; les A_{ki} $(i = k+1, p-1)$, les B_{ki} $(i = 0, k-1)$ et B_{ik} $(i = 0, p-1$, tel que $i \neq k)$. Et finalement, la troisième partie est constituée de $(p-1)^2$ triadiques sur les blocs ; dès qu'un A_{ki} est calculé on peut en déduire les A_{ij} $(i = 0, p-1$, $j = k+1, p-1$, et tel que $i \neq k)$ et les B_{ij} $(i = 0, p-1$ et $j = 0, k-1$, et tel que $i \neq k)$. L'ensemble de la méthode se base sur 3 tâches matricielles, soit :

- p inversions de matrice.
- $2p(p-1)$ produits de matrices.
- $p(p-1)^2$ triadiques matricielles.

Il n'y a pas d'opérations d'affectation de blocs. L'implémentation de ces opérations utilise les bibliothèques LAPACK [ABB+] pour l'inversion et BLAS3 pour le produit et la triadique matricielle.

Inversion de matrice par Gauss-Jordan par blocs

Entrée : A (partitioné en $p \times p$ blocs)

Sortie : $B = A^{-1}$

Pour $k = 0, p-1$

$B_{kk} = A_{kk}^{-1}$

Pour $i = k+1, p-1$ ...**(1)**

$A_{ki} = B_{kk} \times A_{ki}$

Fin Pour

Pour $i = 0, p-1$ ...**(2)**

Si $(i \neq k)$ $B_{ik} = -A_{ik} \times B_{kk}$

Si $(i < k)$ $B_{ki} = B_{kk} \times B_{ki}$

Fin Pour

Pour $i = 0, p-1$ ...**(3)**

Si $(i \neq k)$

Pour $j = k+1, p-1$

$A_{ij} = A_{ij} - A_{ik} \times A_{kj}$

Fin Pour

Pour $j = 0, k-1$

$$B_{ij} = B_{ij} - A_{ik} \times B_{kj}$$

Fin Pour

Fin Si

Fin Pour

Fin Pour

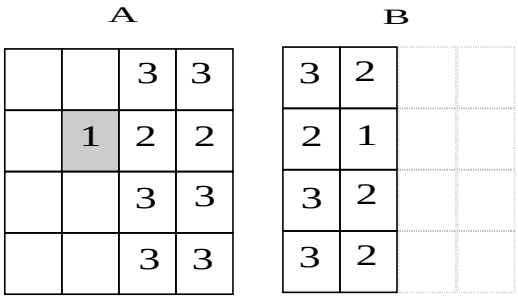


FIG. 5.6 – Dépendances intra-étape.

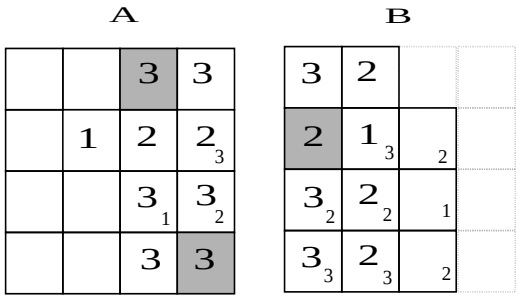


FIG. 5.7 – Dépendances inter-étapes.

Les dépendances dans le code sont de deux types : intra-étapes et inter-étapes. Les figures 5.6 et 5.7 montrent ces dépendances. Les graphes de tâches pour une étape donnée sont montrés sur les figures 5.8 (pour $p = 3$) et 5.9 (dans les cas général). À une étape k, l'inversion de bloc est l'opération la plus bloquante. En effet, dès que celle-ci est faite, les boucles **'Pour'** peuvent être exécutées en parallèle, mais

en respectant les intra-dépendances. Cela veut dire qu'au moins tous les produits matriciels dans **(1)** et **(2)** sont exécutés en parallèle, ainsi que les triadiques matricielles dans **(3)**. Cependant, la boucle **(3)** peut commencer sans que les boucles **(1)** et **(2)** soient totalement finies. L'étape $(k+1)$ peut donc commencer avant la fin de l'étape k. En effet, dès que le bloc suivant sur la diagonale est mis-à-jour, on procède à son inversion. Cette opération devient donc prioritaire. Sur la figure 5.7, l'étape 3 commence avant la fin de l'étape 2. Le calcul sur les blocs représentés par des carrés grisés n'est pas encore fini mais le calcul sur l'étape suivante, représenté par les petits nombre, a déjà commencé. Nos implémentations exploitent le parallélisme intra-étape.

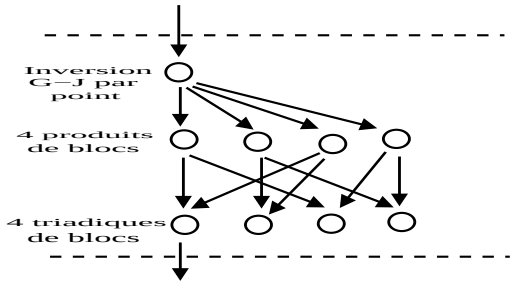


FIG. 5.8 – Graphe de tâches pour $p = 3$.

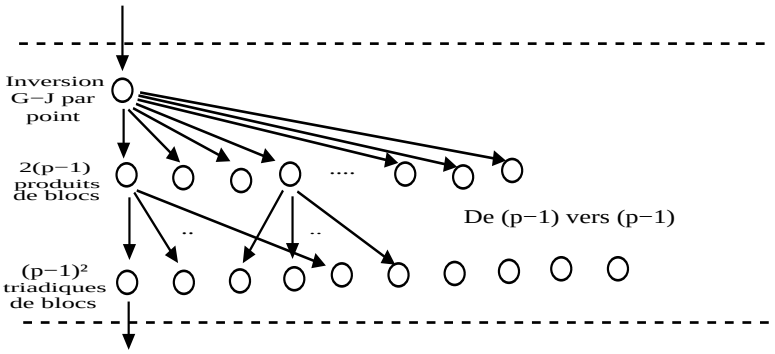


FIG. 5.9 – Graphe de tâches pour une étape donnée.

5.2.2.1 Déploiements sur la grille

L'imlémentation d'un tel algorithme en utilisant XtremWeb n'est pas triviale et s'est avérée être une tâche relativement compliquée. En effet, ce middleware ne gère pas les dépendances et les synchronisations, ni la réutilisation des fichiers résultats intermédiaires. Il a aussi des difficultés à relancer les tâches après des fautes. Dans la plupart des cas, ceci a été géré explicitement comme expliqué plus haut pour le produit matrice-vecteur. L'annexe A donne un extrait de code source client traitant la distribution des boucles (1) et (2) de l'algorithme décrit plus haut. Il faut aussi noter le caractère centralisé des communications et le manque de référencement pour le contrôle et placement des données et tâches.

Il y a aussi des limites sur la taille maximale des fichiers de données envoyés, liées à l'implémentation d'XtremWeb. En effet, XtremWeb devrait utiliser plusieurs protocoles pour le transfert de données (selon la description du RPC multiprotocoles [Fed03] évoqué dans le chapitre précédent). Néanmoins, actuellement ceci repose essentiellement sur Java RMI inadapté aux transferts de flots de données de grande taille. La description XwIDL (basé sur XML) devrait régler ce problème en permettant notamment un choix explicite par l'utilisateur. L'autre problème est la gestion mémoire, liée à Java et son *'Garbage Collector'* qui n'est absolument pas déterminable et surtout n'assure pas que le programme client Java ne soit pas à un moment donné face à une insuffisance de ressources mémoire (*'out of memory'*). Ce qui de facto limite le nombre de boucles dans le programme, et donc le choix des découpes et tailles des matrices.

Les implémentations comportent un total de p^3 tâches matricielles. Pour les communications, nous avons implémenté deux versions. La première gère toutes les communications ; tous les blocs (de la matrice initiale et ceux construits de la matrice cible) sont envoyé sur la plateforme de calcul (en transitant par le coordinateur). Dans la deuxième, on implémente un schéma utilisant le stockage persistant des blocs. Ainsi, dans les boucles **(1)**, **(2)** et **(3)**, seulement un bloc de données est envoyé, les autres sont pré-déployés ou référencés, en l'occurrence générés sur les noeuds de calcul étant donné qu'XtremWeb ne gère pas la localité des blocs de données. Cela veut dire qu'à chaque étape, on simule l'anticipation de l'envoie des blocs sur les neouds où ils seront utilisés, en respectant les dépendances entre tâches, discutées plus haut. Ainsi, pour la boucle **(1)**, le bloc A_{ki} est pré-déployé et dès que le calcul de l'inverse du bloc A_{kk} (c'est-à-dire B_{kk}), on l'envoie. Pour la boucle **(2)**, les blocs A_{ik} et B_{ki} sont pré-déployés. Pour la boucle **(3)**, les blocs A_{kj} et B_{kj} sont cloués sur les noeuds où ils étaient calculés et on anticipe l'envoie de A_{ik}.

La plateforme de calcul est composée de deux sites ; Lille (France) et Tsukuba (Japon). La description des ressources utilisées est donnée en chapitre 4. Le coor-

dinateur est une machine dédié[1] situé à l'École Polytechnique Universitaire de Lille et la bande passante utilisé est de 1.9Mb/s entre le coordinateur à Lille et le site distant. Les matrices sont denses et générales de tailles $N = 3000$ à $N = 12000$, avec taille de bloc $n = 1500$. À chaque exécution, nous avons besoin, au minimum, de $(p-1)^2$ noeuds de calcul disponibles ($p = \frac{N}{n}$). Le programme d'inversion, le produit de matrice et la triadique matricielle sont écrit en C et utilisent les routines `DGETRF` et `DGETRI` de LAPACK (inversion) et la routine `DGEMM` de BLAS3 (produit et triadique). Ces binaires sont liés statiquement (toutes les machines ne contiennent pas les bibliothèques scientifiques LAPACK et BLAS, et pour éviter les problèmes liés aux versions différentes de bibliothèques C sur les noeuds de calcul). Le tableau 5.4 et la figure 5.10 présentent les temps d'exécution et le nombre de tâches matricielles lancées sur la plateforme de calcul à chaque test.

Tailles des matrices	Avec gestion de toutes les communications		En utilisant le stockage persistant		Nombre de tâches matricielles
	moyenne	minimum	moyenne	minimum	
3000	608 s	513 s	411 s	347s	8
6000	3675 s	2765 s	1169 s	977 s	64
9000	6770 s	4453 s	2776 s	2261 s	216
12000	26818 s	17555 s	5057 s	4316 s	512

TAB. 5.4 – Temps d'exécution et nombre de tâches matricielles (blocs de taille 1500).

Ces résultats mettent en évidence l'intérêt du stockage persistant pour ce type d'algorithme sur ces plateformes. Un facteur de 1.4 à 3.5 entre les temps d'exécution des deux implémentations est constaté. Un schéma de placement efficace des blocs est donc nécessaire. Le taux d'anticipation est toutefois supposé maximal dans cette implémentation. Ceci est tout à fait réaliste à condition d'un ordonnancement efficace et un bon choix de p permettant aux temps de calcul de recouvrir cette anticipation. Dans la section suivante, on présente des implémentations à passage de messages, utilisant la bibliothèque MPICH.

5.2.2.2 Versions à passage de messages

Afin de comparer les résultats obtenus avec la plateforme de calcul en Grid, nous avons implémenté des versions à passage de messages sur l'ensemble des clusters de PCs à Tsukuba et le LAN Ethernet à Lille. L'implémentation MPI utilisée est MPICH [GD96][GL04].

Cette version de l'inversion est implémenté comme suit : à chaque étape, le bloc inverse est calculé par le processus maître ; ensuite les boucles **(1)** et **(2)** sont exécu-

[1] http ://lavezzi.rech-info.yser.net/XtremWeb/

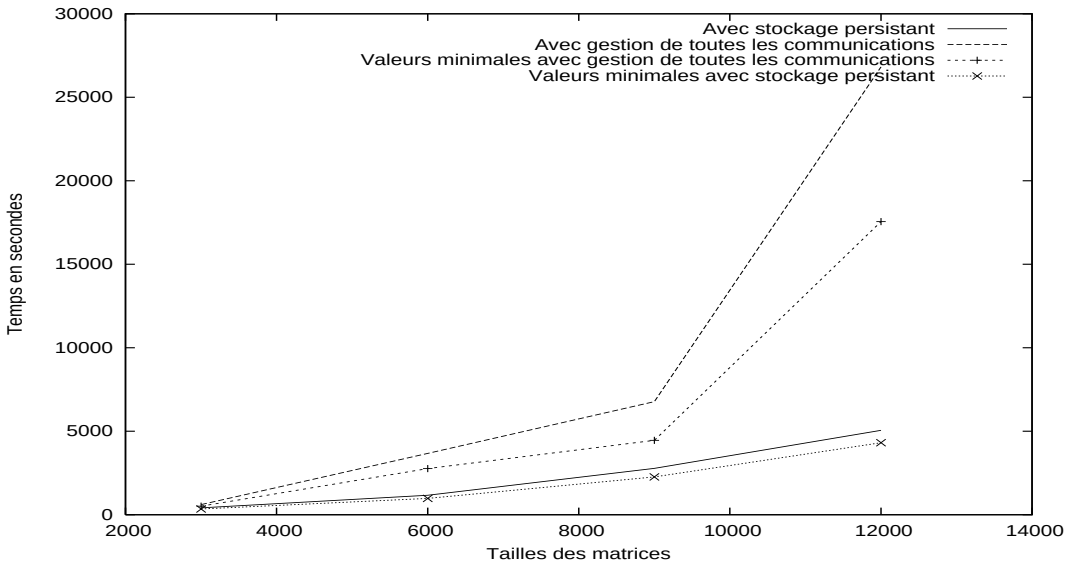


FIG. 5.10 – Inversion de matrice par Gauss-Jordan sur la grille étendu (162 noeuds).

tées en parallèle sur $(p-1)$ processeurs ; puis les triadiques matricielles de la boucle **(3)** sur $(p-1)\times(p-1)$ processeurs. Le processus maître envoie les blocs de données à tous les processus du groupe et chacun exécute des opérations de réception, de calcul (produit ou triadique) et renvoie les blocs résultats au processus maître. Une partition locale sur le disque du maître est utilisée pour le stockage des blocs des matrices (initiales de A et générés de B).

Tailles des matrices	Sur les clusters		Sur le LAN	
	moyenne	minimum	moyenne	minimum
3000	491 s	173 s	262 s	234 s
6000	2255 s	875 s	1364 s	1236 s
9000	5784 s	5592 s	3906 s	3762 s
12000	11792 s	10976 s	7764 s	7203 s

TAB. 5.5 – Temps d'exécution des implémentations à passage de messages, blocs de taille 1500.

Le tableau 5.5 et la figure 5.11 donnent les temps d'exécution des implémentations MPI. Les tests ont été menés sur des matrices de même tailles, générales et denses, et avec la même taille des blocs que les implémentations sur la grille. Deux séries d'expériences ont été réalisées. La première sur la plateforme à Lille sur un réseau local Ethernet et la deuxième sur l'ensemble des trois clusters à Tsukuba. En comparaison avec les résultats sur la grille avec gestion de toutes les communi-

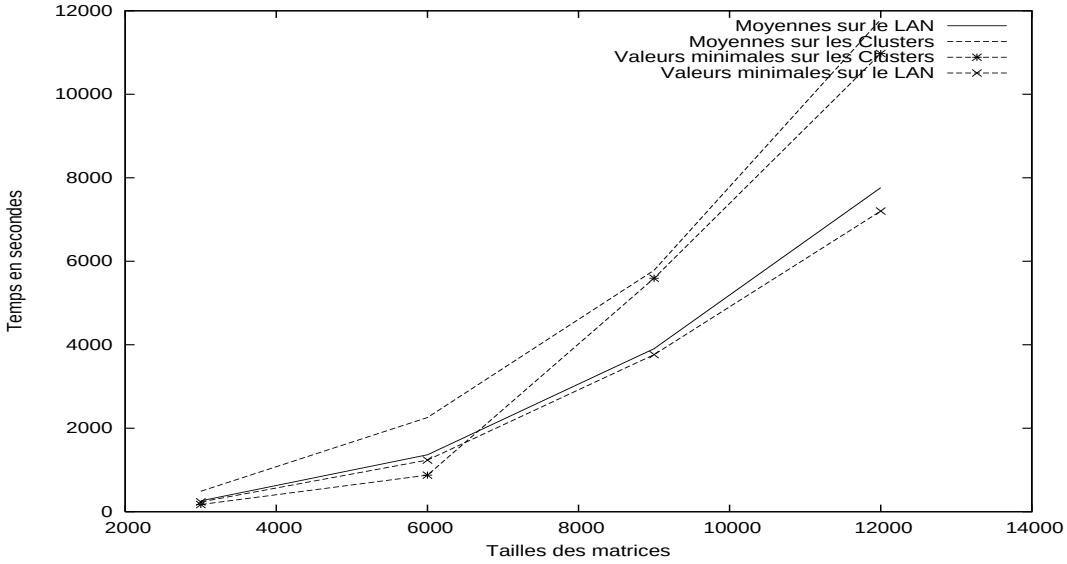


FIG. 5.11 – Implémentations à passage de messages de l'inversion par Gauss-Jordan sur le LAN et les clusters.

cations, les implémentations à passage de messages sont meilleures par un facteur de 1.2 à 3.4. Néanmoins, le stockage persistant des données sous XtremWeb offre de meilleurs temps de calcul dans 75% des cas présenté sur les tableaux 5.5 et 5.4 par un facteur jusqu'à 2.5 (pour la taille de matrice 12000). Un schéma de placement persistant est donc requis afin d'atteindre les performances des implémentation classiques à passage de messages en environnements locaux.

Par ailleurs, la qualité d'interconnexion des clusters étant relativement élevée, on s'attendait à de meilleurs temps de calcul. Cependant, les résultats montrent que la moyenne sur le LAN est meilleure. Ceci est dû à une charge de travail élevée, des insuffisances mémoire et paginations fréquentes, et une plus grande latence sur les clusters au moment des tests. Les raisons de cette latence anormalement élevée peuvent être multiples ; les goulots d'étranglements peuvent se produire dans n'importe quel composant sur le chemin du flux des données, en allant des systèmes d'exploitation sur les noeuds, jusqu'au réseau de communication (avec des liens défectueux par exemple). À noter aussi l'absence de système de réservation de noeuds (*batch*) sur les clusters. Les résultats obtenus montrent ainsi le problème des limites et contraintes mémoire sur les noeuds de calcul partagés et manipulant de grands volumes de données. Nous discutons cela dans le chapitre suivant en utilisant la programmation *out-of-core* afin de gérer les restrictions mémoire sur les noeuds de calcul. En effet, les noeuds du cluster sont des bi-processeurs et quadri-processeurs et partagent de la mémoire. Deux à quatre tâches matricielles partagent donc la mémoire du noeud de calcul et ceci génère plus d'activité disque et affecte négati-

vement les performances.

Par ailleurs, sur les noeuds d'un même cluster, les temps de calcul sont meilleurs que le LAN et l'ensemble des clusters. Les temps de calcul minimum pour les tailles de matrice 3000 et 6000 sont 173 secondes et 875 secondes (tableau 5.5). Ceci est justifié par le fait que les communications entre les noeuds d'un même cluster sont 10 fois plus rapides qu'entre noeuds de différents clusters (941Mb/s contre 94Mb/s). Les autres tailles des matrices nécessitent plus de processeurs que dans un seul cluster et ceci augmente les temps de calcul. La vitesse des communications est donc très importante dans les performances des applications matricielles distribuées. Seulement, les résultats montrés aux paragraphes précédents nous font constater que la quantité mémoire disponible et la charge des noeuds de calcul est un élément de performance aussi déterminant que les communications et qu'il est préférable de faire des communications plus lentes vers des machines plus disponible que de communiquer rapidement des tâches à des machines avec des limites de taille mémoire et des charges moyennes élevées (qui peuvent être dû à plusieurs paramètres et notamment de configuration).

En effet, d'autres tests sous XtremWeb pour le produit matrice-vecteur et l'inversion matricielle (de tailles 10000 et 3000 respectivement) ont montré d'abord une amélioration des temps de calcul moyens de l'ordre de 16% pour le produit (229 secondes sur le cluster et 272 secondes sur la plateforme locale), et 30% pour l'inversion (1110 secondes contre 1565 secondes). Néanmoins, cette tendance n'est pas confirmé dans les tests suivants et les performances des clusters chutent de 20% en moyenne (pour les mêmes tailles). Nous avons mentionné plus haut dans cette section que les goulots d'étranglements peuvent se produire à n'importe quel niveau sur le chemin des données. Néanmoins, cette deuxième série de tests confirme la nécessité de gérer les contraintes liées à la taille mémoire disponible pour l'application sur les noeuds de calcul et ceci indépendamment de la qualité d'interconnexion. C'est l'objet du chapitre suivant.

5.2.3 Résultats sur Grid'5000

Les tests sur Grid'5000 ont été menés essentiellement sur les noeuds d'Orsay (Grid eXplorer) et des noeuds à Sophia Antipolis, Grenoble et Rennes. Sur ces quatres sites, trois ont des bi-processeurs AMD-64 opteron à 2 et 2.2 GHz, et 2GB de mémoire vive, et le quatrième (Grenoble) des bi-processeurs Intel Xeon à 2.4GHz et 1.5GB. Les bandes passantes et latences moyenne utilisées entre le coordinateur (situé à Orsay) et les noeuds de calcul sur les clusters sont données sur le tableau 5.6. Les noeuds déployés (jusqu'à 150 sur les quatres sites) sont dédiés à ces tests. Les tableaux 5.7 et 5.8, et la figure 5.12 montrent les résultats des versions par blocs du produit matrice-vecteur et de l'inversion par Gauss-Jordan.

	bande passate	latence
Orsay	941 Mb/s	0.09 ms
Orsay - Sophia	13.65 Mb/s	18 ms
Orsay - Grenoble	20.87 Mb/s	12 ms
Orsay - Rennes	27.25 Mb/s	9 ms

TAB. 5.6 – Bandes passantes et latences moyennes entre le coordinateur et les sites de calcul.

Tailles des blocs	Avec gestion de toutes les communications		En utilisant le stockage persistant	
	moyenne	minimum	moyenne	minimum
1500	44229 ms	38342 ms	36470 ms	31752 ms
3000	37865 ms	33198 ms	23470 ms	14622 ms
3750	41845 ms	32844 ms	22041 ms	15461 ms
300-par-30000	30613 ms	22290 ms	13267 ms	8749 ms

TAB. 5.7 – Temps d'exécution sur les noeuds de Grid'5000.

Tailles des matrices	Avec gestion de toutes les communications		En utilisant le stockage persistant	
	moyenne	minimum	moyenne	minimum
3000	327 s	297 s	166 s	143 s
6000	1137 s	1107 s	423 s	330 s
9000	3860 s	3604 s	659 s	571 s
12000	10301 s	9087 s	1048 s	920 s

TAB. 5.8 – Temps d'exécution (blocs de taille 1500) sur les noeuds de Grid'5000.

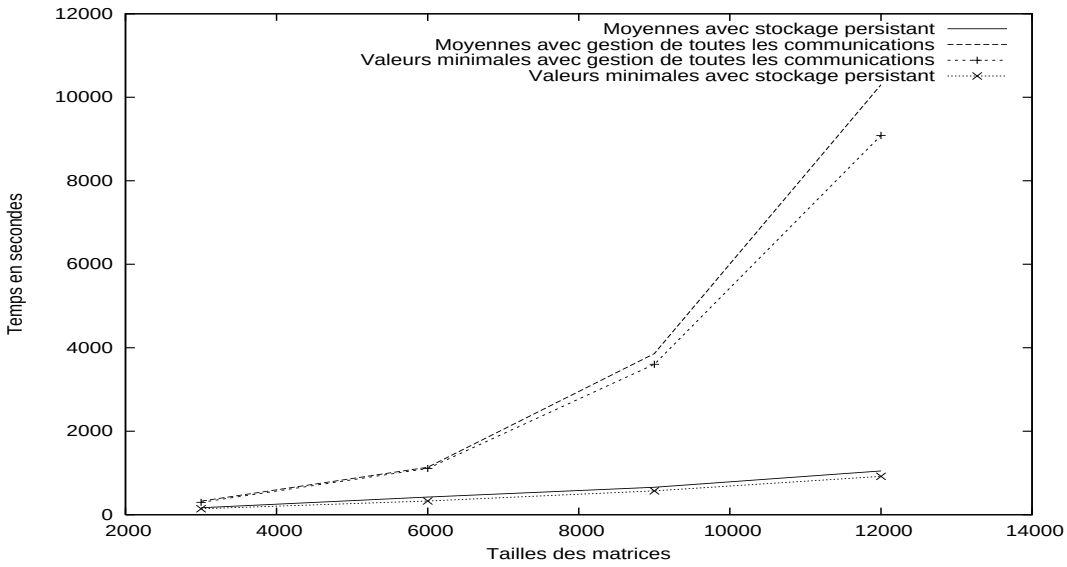


FIG. 5.12 – Inversion de matrice par Gauss-Jordan sur les noeuds de Grid'5000.

En comparaison avec les performances des sections précédentes, les temps de calcul sont meilleurs par un facteur de 6 à 25 pour les versions qui gèrent toutes les communications, et un facteur de 4 à 18 pour les versions avec stockage persistant des blocs de données. Le niveau des performances est donc largement meilleur et cela était prévisible. Nous utiliserons notamment ces résultats en comparaison avec les versions utilisant une gestion explicite des contraintes mémoire sur les plateformes non-dédiées introduite au chapitre suivant. Nous constatons aussi que le stockage persistant donne de meilleurs résultats par un facteur de 1.9 à 9.8. Même avec une meilleure qualité d'interconnexion le placement persistant des données permet d'avoir des gains de performances significatifs. Par ailleurs, les figures 5.13 et 5.14 montrent que les variations des temps de calcul sont parfois importantes malgré le fait que les machines soient dédiées et le système fermé. Ceci est cependant moins vrai pour les petites tailles de matrices.

5.3 Discussion et conclusion

Dans ce chapitre, nous avons montré que l'exploitation de la localité des données est la clé pour concevoir des programmes efficaces sur les grilles de calcul, notamment hétérogènes et non-dédiées. La mise en place de schémas de placements effectifs pour les problèmes d'algèbre matricielle est assez simple à mettre en oeuvre en présence d'une gestion de références sur les blocs de données dans l'infrastructure de programmation utilisée. Il est évident que cette gestion génére un surcoût qu'il faudra couvrir par de bons choix de découpage et distribution avec éventuellement des techniques de re-distribution des blocs de données. Cependant, il n'est pas

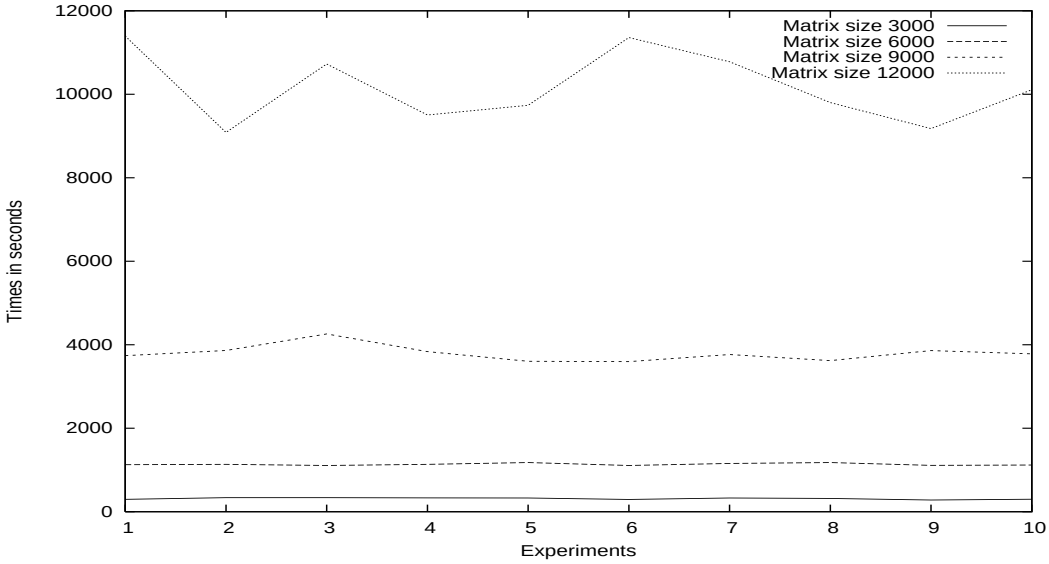


FIG. 5.13 – Variation des temps de calcul sur dix exécutions avec gestion des communications.

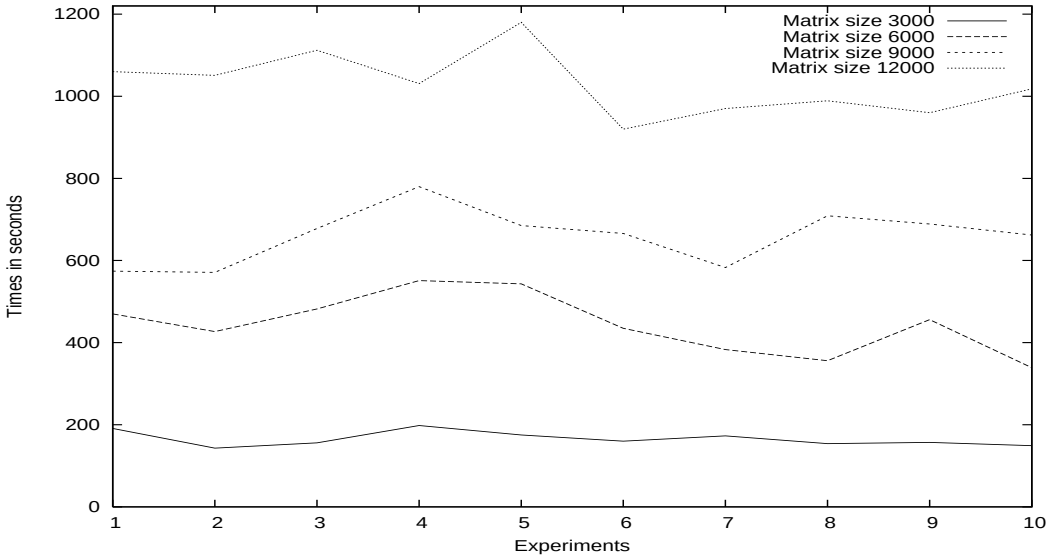


FIG. 5.14 – Variation des temps de calcul sur dix exécutions avec stockage persistant.

toujours souhaitable de réaliser dynamiquement des migrations de données car ces mouvements peuvent devenir incessants et engendrer plus de communications que la quantité totale générée par l'algorithme. Par ailleurs, ce problème est NP-complet même pour des cas très simples d'hétérogénéité (présente uniquement dans les fréquences CPU ou les temps de communication par exemple). Nos résultats montrent donc qu'un schéma initial et préalable de distribution et placement de données (accessoirement de tâches) est nécessaire pour le calcul matriciel sur la grille.

D'un autre côté, il y a deux sources potentielles pour la localité des données : le réseau, exploité dans ce chapitre en utilisant un placement persistant des blocs de matrices afin de réduire les communications, et la mémoire. Pour cette dernière, et de façon similaire au placement persistant, nous voulons optimiser l'utilisation des données à un niveau inférieur en mémoire avant de les déplacer au niveau supérieur. Les résultats de ce chapitre, notamment de la version à passage de messages, ont clairement montré la nécessité de gérer explicitement les contraintes mémoire étant donné que la pagination (ou *swapping*) dégrade sérieusement les performances. Par ailleurs, le décalage entre l'accroissement des performances des processeurs et les débits des réseaux d'un côté et des mémoires et disques de l'autre, définit les entrées/sorties disques comme le goulot d'étranglement principal. En effet, les performances des processeurs et des réseaux de communication ont augmenté de 30% à 50% par an ces dernières années, alors que les performances des mémoires et disques que de 3% à 5% ! L'écart de performance peut devenir encore plus grand dans les années à venir. Le chapitre suivant traite cette problématique en proposant une gestion *out-of-core* des tâches matricielles de base pour le calcul sur les grilles non-dédiées.

Chapitre 6

Paradigme de programmation à grande échelle

6.1 Introduction

Les paradigmes de calcul maître-esclave ou multi-paramètres inhérents aux systèmes de grille et pair-à-pair actuels ne sont pas nécessairement adaptés à l'algèbre linéaire, notamment en raison des communications (granularité du calcul), et aussi de l'hétérogénéité de la plateforme. En effet, les machines dans le réseau peuvent être hétérogène à differents niveaux ; en particulier les processeurs, avec différentes fréquences ; le réseau, avec différents coûts de communication ; et la mémoire, avec différentes capacités et hiérarchies mémoire sur les différentes machines. Dans ce chapitre nous étudierons l'impact de l'hétérogénéité mémoire sur les performances des programmes distribués sur la grille et nous nous intéresserons à une extension de paradigme de calcul traitant les restrictions mémoire sur les noeuds de calcul.

Il est évident que la taille des données est limitée par la mémoire présente sur les noeuds de la grille. Plusieurs problèmes intéressants apparaissent quand on veut atteindre les limites du système, car en général nous devons faire attention à ne pas excéder la capacité mémoire des machines lors de l'allocation des tâches. Ceci détermine aussi la taille des données en entrée pour nos programmes. Une manière de déterminer la mémoire disponible est d'utiliser la valeur de la mémoire physique disponible pour les applications en se basant sur la valeur de la mémoire physique disponible dans le système. Ceci est inadapté pour les grilles de calcul car les machines sont généralement non-dédiées et cela doit être fait dynamiquement. Cependant, nous pouvons nous baser sur cette valeur pour décider d'une taille maximale et pour paramétrer nos programmes, c'est-à-dire utiliser des algorithmes particuliers de replacement de pages mémoires en se basant sur un paramètre qu'on peut qualifier de 'Taille de Mémoire Résidente' qui est définit comme le nombre minimum de pages mémoire assurant que tous les défauts de pages ne sont dûs qu'à la première référence. Il faut aussi noter que cela dépend de l'application comme nous allons le

voir dans la suite de ce chapitre. Nous proposons un paradigme de programmation sur la grille basé sur la programmation *out-of-core* et présentons une modélisation avec des hypothèses pour les applications testées.

6.2 Problèmes relatifs au calcul en environnements hétérogènes

Dans cette section nous rappelons les problèmes relatifs au calcul parallèle et distribué en environnements hétérogènes, en commençant par ce qui nous intéresse en premier, les contraintes mémoire. Cela nous permettra de définir les paramètres potentiels utilisable dans un modèle de calcul sur la grille.

- **Mémoire** : la quantité de mémoire physique (et donc de mémoire allouée aux applications) est différente sur l'ensemble des machines de la grille. Ceci est aussi très important quant à la décision de la taille du problème la plus large qu'on peut atteindre dans des conditions normales d'exécution. Nous proposons pour cela une approche de programmation *out-of-core* pour le traitement de ces contraintes et la gestion de la scalabilité.

- **Latence et bande passante réseau** : ces paramètres sont très importants en environnements hétérogènes. Une grande latence peut s'avérer très coûteuse en communications. Aussi l'augmentation du nombre des machines dans la grille sous ces conditions restreint la scalabilité, rendant ainsi la puissance de calcul et le parallélisme inopérant. Il est donc crucial de réduire la latence en développant des réseaux rapides. Cependant, tant que les processeurs deviennent de plus en plus rapide (la fameuse loi de Moore), la latence continuera à être un goulot d'étranglement. La largeur de bande est aussi un goulot d'étranglement, et en dépit de la croissance des largeurs de bande physiques, le débit utile pour les applications est une petite fraction de ce dernier (notamment sur les réseaux locaux et en fonction des protocoles) [ZLC95]. Avec différentes interconnexions, l'hétérogénéité réseau est donc un facteur significatif en calcul parallèle et distribué sur la grille.

- **Problème de logiciel et contraintes des données** : la différence en OS, système de fichiers, bibliothèques et compilateurs disponibles, doit être masquée dans les grilles informatiques. Aussi, l'hétérogénéité des machines entraîne différentes représentations des données. Ceci peut engendrer des coûts de conversion. Il existe aussi des standards pour la présentation des données tel que XDR (eXternal DATA Representation) courant pour les nombres à virgule flottante.

- **Ordonnancement et équilibrage de charge** : typiquement un large nombre de machines est disponible et nous devons en sélectionner une ou un sous-ensemble optimal. L'attribution de tâches à chaque processeur doit se faire en fonction de ces caractéristiques. Nous devons aussi considérer l'hétérogénéité du réseau lorsqu'on place les tâches et les communications. Un problème d'équilibrage de charge en découle et les algorithmes homogènes d'équilibrage doivent donc être adaptés à fonctionner en environnements hétérogènes. Un équilibrage de charge dynamique (à l'exécution) peut aussi être requis suivant le changement d'état de la plateforme. Plusieurs schémas et outils d'équilibrage de charge, dynamiques ou statiques, ont été proposés pour applications parallèles en environnements hétérogènes [KS05][SK04][LC][Leg03]. Néanmoins, ce problème conduit souvent à des résultats de NP-complétude même pour des applications simples et des cas élémentaires d'hétérogénéité.

- **Parallélisation et paradigme de programmation** : les applications parallèles entrent dans un certain nombre de catégories et peuvent avoir des calculs et communications réguliers ou irréguliers. Aussi, l'hétérogénéité des ressources et toutes les caractéristiques inhérentes aux grilles de calcul rendent les modèles classiques inadaptés. D'un autre coté, la décomposition et le placement hétérogène des tâches peut aussi avoir un effet drastique sur la performance. Les problèmes des applications avec un grand rapport communication/calcul sur ces environnements nous amène à nous interroger sur la validité de faire ces calculs sur des réseaux qui souffrent, malgré toutes les avancées technologiques, d'une grande latence et d'une largeur de bande utile assez restreinte. Cela dépend aussi des outils et protocoles utilisés. De ce fait, la définition de nouveaux paradigmes de programmation et l'optimisation des communications est, de notre point de vue, d'importance primordiale dans l'avenir du calcul scientifique sur la grille.

6.3 Calcul out-of-core

Dans la pratique, la taille de la mémoire est le facteur fixant la taille maximale d'un problème. Cependant, le paradoxe entre, d'un coté des mémoires très rapides mais de petites tailles et coûteuses, et de l'autre des mémoires bon marché, de grandes tailles mais très lentes d'accès, est présent et se pose comme un handicap car on considérera qu'il est inconcevable de recourir à la pagination du fait de l'importante chute des performances. Aussi, dans l'absence d'une augmentation substantielle du rapport mémoire/puissance de calcul, il est normal et inévitable de développer des solutions *out-of-core* pour les grands problèmes d'algèbre linéaire. En effet, l'accroissement de la puissance des processeurs, mais aussi des débits réseau, définit les entrées/sorties disques comme le facteur le plus pénalisant dans le cadre des applications matricielles distribuées à grande échelle.

La problématique du calcul *out-of-core* n'est pas nouvelle. Beaucoup d'études avaient introduit cette notion par le passé [Tol99][DHW97][Rei][CU][KBC+03][BCR96]. Le concept est d'utiliser au mieux les ressources des mémoires externes. L'approche la plus efficace est algorithmique, c'est-à-dire en redéfinissant les codes. Il existe aussi une approche générique via la compilation sans réécriture de code [MDK]. Cependant, comme tout outil de compilation, les performances restent loin d'un traitement spécifique. Les algorithmes étudiés par la suite seront basés sur l'approche algorithmique.

Sur grilles de calcul, les machines sont hétérogènes et non-dédiées, donc le manque en mémoire vive face aux applications scientifiques constituera un obstacle fortement contraignant. Nous nous situons dans un cadre de traitements *out-of-core* spécifiques sur chaque noeud, c'est-à-dire que la gestion des accès mémoire est local. Les calculs sont donc ordonnancés de façon à réduire au maximum le nombre des accès disques et/ou de recouvrir calcul et temps de communication.

Les architectures actuelles des stations de travail renferment plusieurs niveaux mémoire, dont l'ordre est déterminé par le temps d'accès moyen. Généralement, le volume de stockage est inversement proportionnel au temps d'accès. La figure 6.1 donne une représentation de la hiérarchie mémoire : registres, differents niveaux de cache, mémoire vive et disque. Dans nos implémentations nous considérerons les deux derniers étages de cette hiérarchie, la mémoire vive et le disque.

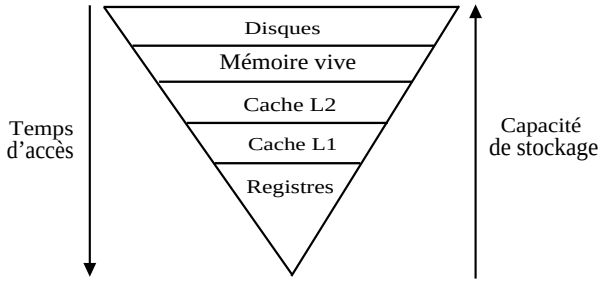


FIG. 6.1 – Représentation de la hiérarchie mémoire.

6.3.1 Aspects système

Dans cette section, nous rappelons les mécanismes de gestion de la mémoire virtuelle sous Linux (sur lequel tourne toutes les machines utilisées dans nos expérimentations). Lorsque les ressources mémoire nécessaires à un calcul dépassent la capacité disponible, c'est ce mécanisme, appelé aussi pagination, qui permet de poursuivre le traitement.

6.3.1.1 La pagination

L'organisation de la mémoire sous Linux repose sur le fonctionnement par pagination. Rappelons qu'un processus correspond à deux données : une configuration de son espace d'adressage, et un ensemble de threads. Le système maintient une table des pages, en fonction de la configuration des espaces d'adressage courants, contenant diverses informations comme l'adresse physique des pages et les informations de contrôle d'accès. C'est à partir de cette table que le système effectue la traduction entre adresses effectives ou virtuelles, relatives à l'espace d'adressage, et adresses physiques (figure 6.2).

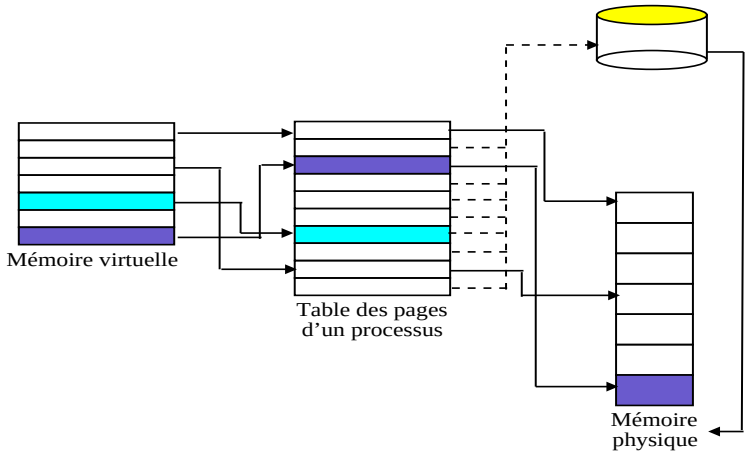


FIG. 6.2 – Pagination mémoire.

Lorsque un processus requiert l'accès à une donnée soit le système dispose d'une adresse physique valide en mémoire et la donnée est transmise au processus, soit l'adresse physique n'est plus valide et la donnée n'est donc plus disponible en mémoire. C'est donc un défaut de page. Dans ce cas, une routine du noyau est exécutée pour résoudre cette erreur (la fonction `fault.c : do_page_fault()`, puis la fonction `memory.c : handle_mm_fault()`). Le comportement de cette routine dépend de la page associée à l'adresse virtuelle, ou du type d'accès effectué. Si la page n'appartient à aucune région ou est de type lecture seule accédée en écriture, le système envoie le signal `SIGSEGV` (le fameux "segmentation fault"), sinon, une page physique est allouée en mémoire et la donnée y est copié.

Le coût excessif engendré par le mécanisme de défaut de page découle implicitement des accès disque. Pour obtenir de bonnes performances de calcul, il est indispensable de minimiser ces accès. L'ensemble est optimal lorsque l'espace de

travail est disponible en mémoire. On cherchera alors à maximiser les accès à une même page par un choix judicieux au niveau algorithmique.

6.3.1.2 Politique de gestion de pagination

On donne ici un aperçu de la politique de gestion de pagination sous Linux. Lorsque le système génère un défaut de page, la page doit être transférée du disque vers la mémoire. Lorsqu'il n'y a plus de page mémoire disponible, le système doit libérer l'espace mémoire nécessaire en déchargeant certaines pages. L'état de la page doit être pris en compte lors de cette opération. Le choix de la page qui va être déchargé de la mémoire influence considérablement les performances. Une stratégie optimale serait de décharger la page mémoire qui serait réutilisé en dernier parmis celles qui se trouve en mémoire. Ceci est imposible à mettre en place dans le cas général puisqu'on ne connait pas les références futures.

La politique de pagination sous Linux est LRU (Least Recently Used). Elle est basée sur le temps : le système associe à chaque page une ancienneté en fonction du dernier accès à celle-ci. Les premières pages à être évincées sont donc les plus anciennes. Il existe d'autres politiques de pagination :

- la politique basée sur une FIFO (First In First Out) : les pages sont déchargées selon leur ordre d'arrivée,
- la politique de la seconde chance : c'est une extension de la politique FIFO. Chaque page positionne un bit, appelé de seconde chance, à 1 lors de son chargement. Lorsqu'une page est candidate au déchargement de la mémoire, selon l'ordre FIFO donc, si son bit de la seconde chance est à 1 on le met à 0 et on garde la page ; sinon elle est déchargée.
- la politique basée sur la fréquence, appelé LFU (Least Frequently Used) : dans cette politique se sont les pages les moins fréquemment utilisées qui sont évincées en premier. L'inconvénient est qu'une page qui a été fortement utilisée va s'attarder en mémoire alors que sa période d'utilisation est terminée.

6.3.2 Gestion de la mémoire en out-of-core

Les politiques de pagination décrites plus haut sont une solution "*out-of-core*" triviale présente dans les systèmes d'exploitations actuels. Malheureusement, ces politiques sont inefficaces. Dans le système Linux, la politique LRU considère que les pages qui ont le plus de chance d'être accédées par la suite sont celles qui viennent d'être utilisées. Cette politique n'est pas adaptée aux accès linéaire en mémoire qui apparaissent dans bien des calculs générés par des applications numériques. Ceci peut engendrer ce qu'on appelle un *'effet trou'* lorsque la taille des données est plus grande que la mémoire physique, le système de pagination est pris à défaut et un phénomène de défaut de page se répercute à chaque page mémoire accédée.

Pour résoudre ce problème, une solution système peut être envisagée, en modifiant la politique de pagination. Ceci n'est pas du tout aisé. Cela consiste en général à réécrire des parties du noyau Linux qui gère la stratégie de gestion mémoire. Cependant, il existe un outil (*MMUM/MMUSSEL*) qui permet un contrôle de la pagination en mode utilisateur [Coz03]. Une utilisation de cet outil pour réduire la pagination d'une factorisation LU a aussi été présentée [Car00]. Toutefois, la solution la plus efficace à notre avis est la solution algorithmique. L'utilisateur doit réécrire son code en réordonnant l'accès aux données en utilisant des entrées/sorties explicites afin d'éviter toute pagination. Cette approche est décrite dans la section suivante.

6.3.3 Approche algorithmique

La façon la plus efficace de traiter les restrictions mémoire est un ordonnancement intelligent des entrées/sorties au niveau algorithmique, c'est-à-dire que l'ordre des entrées/sorties est choisi de façon à les minimiser. Une disposition des données par blocs est aussi choisie de sorte que les entrées/sorties soient exécutées sur ces grands ensembles et que toutes ou la majeure partie des données lues soient employées avant d'être évincées de la mémoire.

Certains algorithmes sont plus difficile à modifier pour un ordonnancement *out-of-core* que d'autres, dans le sens où les contraintes de flux de données et de contrôle doivent être repensées et changées afin de permettre une programmation *out-of-core* efficace. Il y a aussi matière à discussion sur le coût éventuel, arithmétique et/ou numérique, de telles transformations sur les codes en algèbre linéaire. Pourtant, il y a des exemples très intéressants d'algorithmes numériques (directs et itératives, denses ou creux) qui ont été transformés efficacement [Tol99]. Nous présentons dans la suite un exemple d'approche algorithmique dans une bibliothèque d'algèbre linéaire ; le prototype *out-of-core* de ScaLAPACK, ainsi que nos applications par blocs ; le produit matrice-vecteur et l'inversion de matrice par Gauss-Jordan.

Un exemple : le prototype out-of-core de ScaLAPACK

ScaLAPACK [DHW97] est une bibliothèque parallèle d'algèbre linéaire dédiée aux machines MIMD à mémoire distribuée et clusters de PCs. C'est la suite du projet LAPACK. La figure 6.3 présente la hiérarchie de ScaLAPACK. Elle se base sur les BLAS et leur version parallèle, les PBLAS. PBLAS et ScaLAPACK utilisent les BLACS comme bibliothèque de communication (dont il existe plusieurs implantations au dessus de divers bibliothèques de passage de massages telles que PVM ou MPI). ScaLAPACK offre une extension *out-of-core* permettant d'effectuer différents types de factorisation (LU, QR et Cholesky) sur des matrices de tailles supérieures à la mémoire en utilisant une décomposition de la matrice initiale en sous-matrices carrées de taille suffisamment petite pour tenir en mémoire. Un niveau supplémentaire de décomposition est donc introduit par rapport à la décomposition *in-core*

de ScaLAPACK afin de réduire le volume total d'entrées/sorties requis. Pour cette extension, une interface d'entrées/sorties efficace et portable a été définie, avec une couche de haut niveau qui encapsule les caractéristiques spécifiques à la machine ou à l'architecture (dans un souci de compatibilité avec les codes ScaLAPACK existants) [DD00].

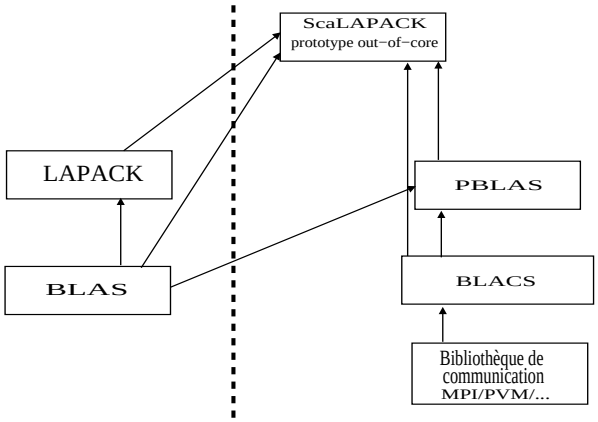


FIG. 6.3 – Hiérarchie de ScaLAPACK.

6.4 Applications

Dans cette section, nous présentons les implémentations *out-of-core* sur la grille du produit matrice-vecteur et de l'inversion matricielle par Gauss-Jordan. Nous décrivons les ordonnancements *out-of-core* des opérations de base de ces applications, et présentons les résultats d'exécution sur la plateforme de calcul sous XtremWeb. Dans ces versions, l'ordre des opérations de calcul est choisi de façon à minimiser les entrées/sorties, de sorte que toutes ou la majeure partie des données présentes en mémoire soient employées avant d'être évincées.

6.4.1 Le produit matrice-vecteur par blocs

Nous allons à présent décrire la version Grid/*out-of-core* du produit matrice-vecteur par bloc. Le produit matrice-vecteur peut être ordonnancé *out-of-core* en utilisant une technique très simple que nous décrivons ici. Supposant que la taille de la mémoire est M, si la matrice est découpé en blocs de taille $p - par - N$, il suffit que $(p*N+N+p)$ éléments tiennent en mémoire, c'est-à-dire, $p \leq (M-N)/(N+1)$. L'algorithme est le suivant :

```
produit matrice-vecteur out-of-core
for i = 1 to N step p
 for j = 1 to N
  multiply(submatrix A_(p-by-N), vector x, subvector b_(p))
 end for
end for
```

Dans cet ordonnancement le nombre de transferts (mémoire-disque) est égal à $\Theta(N)$ alors qu'il est égal à $\Theta(N^2)$ ($4 * N^2$ au maximum) dans un ordonnancement naif (par point) ou si $p > (M - N)/(N + 1)$. En effet, dans chaque itération de la boucle extérieure, le programme `'produit matrice-vecteur out-of-core'` accède à $(p*(N+1)+N)$ éléments (que nous supposons égal à M). Les mots accédés à ce niveau sont lus et écrits une seule fois. Le nombre total de mots transférés est lié au nombre des itérations de la boucle extérieure (N/p) et au nombre de transferts par itération, i.e. égal à $(N/p) \times 2M$, avec $p = (M - N)/(N + 1)$.

Nous avons redéployé le produit matrice-vecteur en utilisant le binaire *out-of-core* sur les noeuds de calcul. La description des plateformes est donnée au chapitre 4.

6.4.1.1 Expérimentations et résultats

Avant de lancer cette version du produit sur la grille de calcul, nous avons exécuté un produit *out-of-core* de taille $N = 30000$ (avec $p = 3000$) sur un ordinateur portable (Pentium 4, 2.4GHz et 512MB). Le temps de calcul moyen est de 907 secondes. Ce résultat est plus intéressant que le produit matrice-vecteur sous XtremWeb avec blocs de taille 1500 (chapitre précédent). Ceci montre qu'il est théoriquement possible de multiplier la taille du problème par le nombre de noeuds dans la plateforme (exécuter un produit de taille supérieure à 3×10^6).

Néanmoins, les caractéristiques des noeuds de calcul sont très importantes dans la gestion des tâches *out-of-core*. L'ordonnancement doit constamment tenir compte de la mémoire disponible allouable et de l'espace disque sur les noeuds de calcul candidates. Pour cette raison, et puisqu'il n'est pas possible de choisir une politique pour l'ordonnancement de tâches (en réalité inexistant dans XtremWeb), nous avons choisi les noeuds de calcul qui participent à ces versions *out-of-core*, en particulier, avec plus de 256MB de mémoire vive et assez d'espace disponible dans les partitions disques allouées.

Le produit *out-of-core* (de taille $N = 30000$) a été exécuté sur la plateforme de calcul sous XtremWeb, avec $p = 1000$ (taille des blocs $1000 - par - 30000$), i.e. 30 tâches de calcul matriciel. Le temps de calcul moyen est 39 secondes, en utilisant le stockage persistent des blocs de la matrice. C'est le meilleur temps de calcul obtenu pour le produit de taille 30000, pour toutes les configurations et versions

considérées (sur les plateformes de calcul non-dédiés). Aussi, en comparaison avec les résultats obtenus sur les noeuds de Grid'5000 (tableau 5.7), ce temps de calcul est à peine plus important par un facteur de 1.3 à 2.9. En outre, par rapport aux résultats obtenus en 5.2.1, l'association du stockage persistant et la gestion explicite des entrées/sorties disque donne des réductions des temps de calcul moyens jusqu'à un facteur 6. Il est donc évident que le traitement explicite des contraintes mémoire sur les noeuds de calcul est pertinent. Aussi, cette version utilise trois fois moins de ressources que la version classique (les tâches matricielles sont plus grandes ; blocs de données de taille $1000 - par - 30000$ au lieu de $300 - par - 30000$).

Nous avons augmenté la taille de la matrice de test pour $N = 300000$ (100 fois la matrice initiale avec blocs générés sur les workers). Le temps de calcul moyen de ce produit (avec $p = 1000$) est de 575 secondes. Ceci est très intéressant car le schéma de calcul traditionnel ne peut pas atteindre ce degré de scalabilité pour cette application. Le traitement *out-of-core* sur les noeuds de calcul permet donc le passage à l'échelle. Ce traitement est en effet un étage supplémentaire dans la gestion de la localité nécessaire aux larges applications matricielles sur la grille. Pour espérer avoir de bonnes performances nous considérons qu'il est indispensable d'optimiser l'emplacement des données à tous les niveaux ; de la hiérarchie mémoire locale au réseau de communication.

Par ailleurs, sur les noeuds de Grid'5000, la version utilisant le traitement *out-of-core* de la version 2 du produit de taille $N = 30000$ donne une moyenne de 35 secondes (avec gestion de toutes les communications) et 10 secondes (avec placement persistant des blocs). Contrairement aux résultats des tests précédents, cette version est en moyenne moins performante que le produit classique avec gestion de toutes les communications (31 secondes, tableau 5.7). Pour la version avec stockage persistant, elle est à peine plus performante en moyenne par un facteur de 1.2. Ceci montre qu'il est pratiquement inutile d'utiliser une gestion *out-of-core* sur ces noeuds dédiés pour les tailles considérées.

6.4.2 L'inversion de matrice par Gauss-Jordan par blocs

L'algorithme de Gauss-Jordan par blocs, décrit au chapitre précédent, est implémenté ici en utilisant un ordonnancement *out-of-core* des tâches matricielles sur les noeuds de calcul. Nous avons écrit des versions utilisants des lectures/écritures disque explicites des trois tâches matricielles nécessaires pour implémenter cet algorithme ; l'inversion matricielle, le produit matriciel et la triadique matricielle.

6.4.2.1 Inversion matricielle out-of-core

Nous implémentons une version *out-of-core* de l'inversion par Gauss-Jordan (section 5.2). Nous introduisons la notion de "sous-bloc" par rapport à l'implémentation

vu au chapitre précédent. Un niveau de décomposition supplémentaire est alors introduit. La mémoire sur les noeuds de calcul est assimilée à ces sous-blocs (et les blocs sont donc associés aux disques), figure 6.4. La mémoire sur chaque noeud pourra contenir au moins trois de ces sous-blocs. Considérons maintenant des sous-blocs de taille $p-par-p$ d'un bloc A^P de taille $P-par-P$. La mémoire principale M peut donc stocker $(3*p^2)$ éléments, p est plus petit que $\sqrt{M/3}$. Le pseudo-code de cette implémentation est le suivant :

```
inversion matricielle out-of-core
for i = 1 to P step p
 inverse_pivot(submatrix A^P_(p-by-p))
 for j = 1 to 2((P/p) − 1)
  multiply(submatrix A^P_(p-by-p), submatrix B^P_(p-by-p))
 for j = 1 to ((P/p) − 1)^2
  triadic(submatrix C^P_(p-by-p), submatrix A^P_(p-by-p), submatrix B^P_(p-by-p))
 end for
end for
```

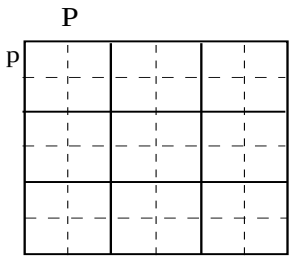


FIG. 6.4 – Distribution en sous-blocs pour l'inversion matricielle *out-of-core*.

6.4.2.2 Produit matriciel et triadique matricielle out-of-core

Pour le produit et la triadique matricielle, on utilise un partitionnement unidimentionnel par ligne pour A, unidimentionel par colonne pour B, et bidimentionnel pour C (figure 6.5). Ces algorithmes sont plus adaptés à un ordonnancement *out-of-core* que l'inversion par Gauss-Jordan. Pour le calcul du produit $A^P \times B^P$ et la triadique $C^P - A^P \times B^P$, les sous-blocs considérés sont des sous-blocs de lignes pour A^P, des sous-blocs de colonnes pour B^P et des sous-blocs carrés pour C^P.

On considère des sous-blocs de taille $p-par-P$ du bloc A^P et de taille $P-par-p$ du bloc B^P. La mémoire sur chaque pair pourra contenir ces deux sous-blocs et un sous-bloc de taille $p-par-p$ de C^P, p est alors plus petit que $\sqrt{P^2+M}-P$. Le

pseudo-code est le suivant :

```
produit matriciel out-of-core
for i = 1 to P step p
 for j = 1 to P step p
  submatrix C^P_(p-by-p) = multiply(submatrix A^P_(p-by-P), submatrix B^P_(P-by-p)).
 end for
end for
```

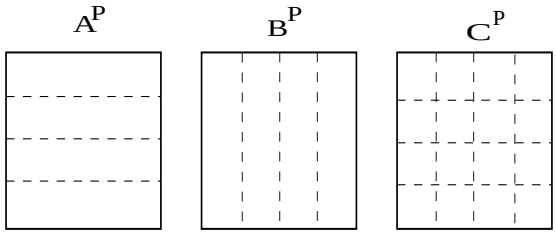


FIG. 6.5 – Distribution en sous-blocs pour le produit matriciel et la triadique matricielle *out-of-core*.

Pour la triadique matricielle, la mémoire contiendra 2 $p-par-p$ sous-blocs de C, le sous-bloc $p-par-P$ de A^P et le sous-bloc $P-par-p$ de B^P, p est plus petit que $\frac{1}{2}\sqrt{P^2+2M}-P$. Le pseudo-code est le suivant :

```
triadique matricielle out-of-core
for i = 1 to P step p
 for j = 1 to P step p
  submatrix C^P_(p-by-p) = triadic(submatrix C^P_(p-by-p),
                                   submatrix A^P_(p-by-P), submatrix B^P_(P-by-p)).
 end for
end for
```

Les routines `inverse_pivot`, `multiply` et `triadic` utilise les bibliothèques LAPACK et BLAS3 et sont liées statiquement. Les binaires à envoyer sont donc plus volumineux mais ceci est nécessaire car les LAPACK et BLAS ne sont pas présents sur tous les noeuds de calcul.

6.4.2.3 Expérimentations et résultats

Les tailles des matrices denses sont de $N = 6000$ à 18000. La taille de bloc considérée est $P = 3000$, et des sous-blocs $p = 1500$. Pour chaque exécution, nous avons besoin d'au moins $((N/P)-1)^2$ noeuds de calcul. Dans ces expériences, toutes les

Tailles des matrices	Avec gestion de toutes les communications		En utilisant le stockage persistent	
	moyenne	minimum	moyenne	minimum
6000	2030 s	1793 s	593 s	496 s
9000	4111 s	3443 s	1617 s	1487 s
12000	5810 s	4910 s	3049 s	2812 s
15000	6991 s	5811 s	4914 s	4609 s
18000	14842 s	13020 s	11700 s	9679 s

TAB. 6.1 – Temps d'exécution de l'implémentation utilisant l'*out-of-core* (blocs de taille 3000 et sous-blocs de taille 1500). La plateforme de calcul est composée des noeuds de Lille et Tsukuba.

communications transitent par le coordinateur, qui lance tous les calculs et contrôle les dépendances et synchronisations.

Le tableau 6.1 et la figure 6.6 montrent les temps d'exécution moyens sur la plateforme de calcul composée du réseau local à Lille et des trois clusters locaux à Tsukuba. En comparant ces résultats avec ceux obtenus dans le chapitre précédent, nous observons une réduction des temps de calcul moyens par un facteur de 1.8 à 6 pour les deux versions ; avec ou sans stockage persistant des blocs des matrices de test. Aussi, le placement persistant augmente les performances dans tous les cas par un facteur de 1.3 à 3.5. Comme pour le produit matrice-vecteur, les versions avec gestion *out-of-core* sur les noeuds de calcul sont largement plus performantes que les versions classiques. Il est important de prendre en compte l'hétérogénéité mémoire qui est un facteur déterminant dans tous les cas de figure, car il n'y a pas de configuration particulière de la plateforme de calcul pour ces tests, notamment concernant le réseau de communication.

Les résultats des tests menés sur les noeuds de Grid'5000 sont donnés dans le tableau 6.2. En comparaison avec les résultats montrés ci-dessus (sur la plateforme de calcul non-dédiée), on gagne logiquement un facteur de 1.8 à 4.2. Par rapport à la version classique sur Grid'5000, on gagne un facteur de 2.3 à 4.4 lorsqu'on gère toutes les communications alors qu'on perd ce gain par un facteur jusqu'à 2.2 dans le cas du stockage persistant. Ceci est très intéressant et montre que le traitement *out-of-core* génère en réalité un *overhead* qui est consommé par la qualité des communications dans le premier cas et ne l'est pas dans le deuxième cas puisqu'il y a moins de communications, et qu'initialement il n'y avait pas de pagination sur les noeuds de calcul dédiés. Le traitement *out-of-core* est par conséquent inopérant pour les tailles de blocs considérées. Pour obtenir de meilleurs résultats, la mémoire vive doit être 'saturée' avec des blocs de données plus importants. Les limites de scalabilité du middleware discutées plus haut dans ce manuscrit, notamment en ce qui concerne

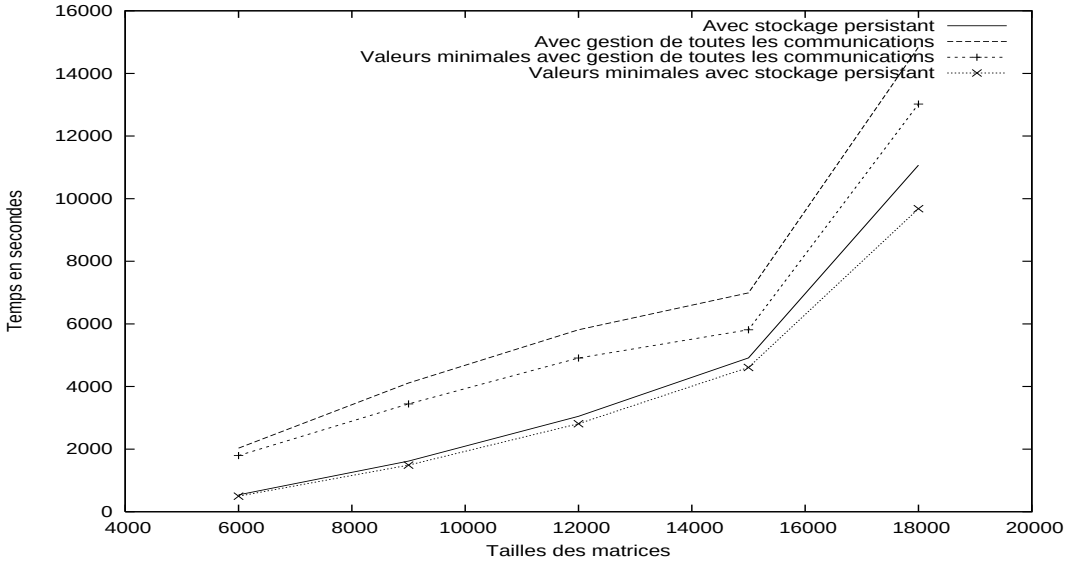


FIG. 6.6 – Graphe des résultats avec valeurs minimums de l'implémentation Grid/*out-of-core* de l'inversion matricielle par Gauss-Jordan.

la taille des fichiers ne permettent pas de tester de plus grandes tailles de matrices et de blocs.

6.5 Modèle GRID/out-of-core

Comme décrit précédemment dans ce chapitre, la solution triviale qui consiste en l'utilisation de la mémoire virtuelle est inefficace, a fortiori dans le cas de l'utilisation de la politique standard de pagination décrite plus haut. Nous avons donc montré que le meilleur moyen pour atteindre de bonnes performances est de restructurer l'algorithme en utilisant des entrées/sorties explicites. Nous traitons ainsi efficacement les restrictions mémoire sur les noeuds de calcul. Dans cette section nous allons présenter un modèle simple associé à nos deux études algorithmiques. Nous parlerons aussi de l'optimisation en abordant le recouvrement des communications et du calcul.

6.5.1 Hypothèses et définitions

Avant d'énoncer nos hypothèses, voici deux définitions :

Définition 1. Une ressource est une caractéristique architecturale qui affecte de manière significative les performances d'un environnement parallèle. Une métrique

Tailles des matrices	Avec gestion de toutes les communications		En utilisant le stockage persistent	
	moyenne	minimum	moyenne	minimum
6000	1112 s	1077 s	1005 s	967 s
9000	1687 s	1611 s	1567 s	1505 s
12000	2320 s	2244 s	2170 s	2137 s
15000	2966 s	2838 s	2728 s	2641 s
18000	3544 s	3398 s	3331 s	3245 s

TAB. 6.2 – Temps d'exécution de l'implémentation utilisant l'*out-of-core* (blocs de taille 3000 et sous-blocs de taille 1500) sur des noeuds de Grid'5000.

ou paramètre est une mesure de cette ressource.

Définition 2. Un modèle est une abstraction de l'environnement de calcul qui est caractérisé par le choix de plusieurs ressources et paramètres correspondants. Un modèle peut alors être identifié par un ensemble de métriques ou paramètres. Les algorithmes seront conçus et analysés à partir de ces paramètres.

Le modèle d'architecture visé consiste en plusieurs ensembles de stations de travail avec disques et hiérarchies mémoire individuelles, et un réseau d'interconnexion hétérogène (composé de plusieurs réseaux d'interconnexions). Chaque noeud stocke ses données (en considérant les blocs comme unités) sur son disque et utilise sa propre mémoire. Les caractéristiques prises en compte concernent les communications, les calculs et les entrées/sorties. Pour des raisons de simplicité, nous considérons seulement les deux derniers niveaux de la hiérarchie mémoire ; la mémoire vive et le disque. Aussi, nous considérons l'ensemble des noeuds comme étant divisé en groupes de clusters ou de machines dans un réseau local, ou un campus par exemple. Ca pourrait aussi être des PCs utilisant des technologies DSL ou câble dans le même secteur géographique, et utilisant probablement le même fournisseur d'accès. Ceci pour poser l'hypothèse de l'existance d'un multicast "fiable" pour notre modèle [HP]. Nous mentionnons qu'il n'existe pas de modèle de calcul approprié aux grilles de calcul qui traite à la fois des communications réseau et des niveaux de mémoire individuels sur chaque noeud (et encore moins d'autres aspects très importants comme la volatilité ou la redondance).

Historiquement, le PRAM (Parallel Random Acces Machine) est le modèle parallèle le plus largement répandu. Cependant, ce modèle est imprécis pour la prédiction du temps réel d'exécution et l'utilisation des ressources, puisqu'il néglige plusieurs détails qui ont un impact important sur la performance, notamment les temps de communications réseau et les problèmes liés à la hiérarchie mémoire et à l'asynchronisme. Plusieurs extentions de ce modèle essayent de le rendre plus pratique tout

en préservant sa simplicité. On peut citer les extentions incorporant des aspects d'asynchronisme (PhasePRAM et APRAM), les coûts de communication comme la latence et la largeur de bande (LPRAM, BSP et LogP), ou la hiérarchie mémoire (P-HMM, PMH et P-UMH) [Li96]. Chacun de ces modèles résume les détails de l'architecture dans plusieurs paramètres génériques qu'on décrira par la suite.

Des paramètres typiques incluent le nombre de processeurs, la latence, la largeur de bande, la topologie du réseau, la hiérarchie et l'organisation mémoire, etc. Les modèles récents essayent d'utiliser plus de paramètres pour capturer le plus finement possible les caractéristiques de l'environnement parallèle. Cependant, un équilibre doit être trouvé afin de ne pas rendre la conception "optimale" d'algorithmes impossible. Le choix de ces paramètres est donc critique pour la conception de modèles utilisables. Nous identifions ainsi les paramètres importants dans notre contexte et les utiliserons pour définir un cadre pour un modèle *out-of-core* sur la grille.

6.5.1.1 Paramètres génériques

Nous rappelons brièvement la liste des ressources et paramètres significatifs , et donc à prendre en compte, pour la définition d'un modèle de calcul sur la grille (figure 6.7) :

- Processeurs ; le nombre de processeurs disponibles et leurs puissances.
- L'organisation mémoire ; hiérarchie et caractéristiques mémoire sur les noeuds (qui peuvent être des stations de travail individuelles ou des machines parallèles).
- Le réseau de communication ; une multitude de paramètres à ce niveau ; la latence, la largeur de bande, les *overheads* liés aux protocoles ou aux architectures des middleware, la topologie, la fiabilité, etc.

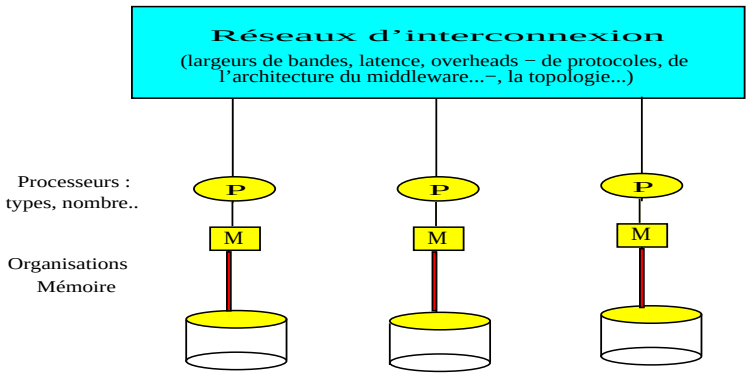


FIG. 6.7 – Hiérarchie des paramètres.

6.5.2 Modélisation

Nous avons décrit les facteurs et hypothèses à prendre en charge dans la définition du cadre de notre modèle. Nous allons à présent définir les constantes et coûts.

Coûts de communication : habituellement, les temps de communication sont représentés par le modèle simple $t_l + L \times t_m$ correspondant au temps de transfert d'un message de taille L avec t_l la latence et t_m le temps d'envoi d'un mot de données. Ce modèle ne prend pas en compte la complexité des réseaux d'interconnexions, car il suppose que la latence et bande passante sont constantes durant le transfert. Le type de communication considéré dans notre modèle est le multicast (noté avec un max). On note t_{comm} le temps de communication d'un élément entre deux noeuds dans la grille, et donc $t_{comm}n^2$ représentra le temps nécessaire à la communication d'un bloc carré de taille n, et $\max_{i=0}^{(p-1)} t_{comm}n^2$ le temps nécessaire à la communication de p blocs carrés de taille n.

Coûts de calcul et des E/S : ces coûts sont naturellement très variables et fonction de plusieurs paramètres ; fréquence et type du processeur, mémoire disponible, hiérarchie mémoire, débit disque et aussi le type du calcul. Nous définissons les constantes suivantes : M_p est l'espace mémoire allouable sur un noeud, c_{io}^p est le coût des routines de lecture/écriture disque d'un élément du bloc sur un noeud p, et les constantes relatives aux types de calcul effectués (i.e. les BLAS et LAPACK). La hiérarchie considéré a été définie dans la section précédente et la localité des données est donc gérée explicitement. Pour nos applications, présentées plus haut, nous avons quatre types de traitement : multiplication matrice-vecteur (`DGEMV`, t_{mv}^p), l'inversion de blocs (`DGETRF`, t_{rf}^p et `DGETRI`, t_{ri}^p), le produit matricielle (`DGEMM`, t_{mm}^p) et la triadique matricielle (`DGEMM`, t_{tr}^p). Nous modéliserons nos applications dans le paragraphe suivant.

6.5.2.1 Conception et d'analyse : exemples

Pour chacune de nos applications, on considère les temps de calcul, des E/S et des communications. Pour le produit matrice-vecteur, soit N la taille du problème. Nous considérons des blocs de taille $p - par - N$, avec $p = (M_p - N)/(N + 1)$, M_p étant la mémoire allouable la moins importante pour le sous-produit sur les noeuds de calcul. On a $B = N/p$ blocs de $(N*p)$ éléments et donc B multiplications par bloc sur B noeuds de calcul. Le coût total correspond au maximum des envoie/réception des blocs de données, et du coûts de la fonction `DGEMV` de taille $p - par - N$, plus la lecture/écriture des blocs de matrice et vecteur. En terme de constantes définies auparavant, le coût total est :

$$T_N = \max_{i=0}^{(B-1)}(t_{comm}(N*(p+1)) + t_{comm}p) + \max_{j=1}^{p}(c_{io}^p(N*(p+1)+p) + t_{mv}^p)$$

Ce coût peut être noté avec des sommes à la place du max en l'absence de l'hypothèse sur le multicast.

Pour l'inversion de matrice par Gauss-Jordan, les contraintes mémoire sont définies comme suit : 3 sous-blocs de taille $p - par - p$ tiennent en mémoire, p est inférieur ou égal à $\sqrt{M_p/3}$. Pour une étape k le coût de communication est : 1) deux fois la diffusion de $(p-1)$ blocs vers $(p-1)$ noeuds de calcul, et 2) diffusion d'un bloc vers $2(p-1)$ neouds de calcul. Le reste des communications est anticipé et est couvert par les calculs (utilisant les règles de placement citées dans le chapitre précédent). Chaque étape contient une inversion, $2(p-1)$ produits matriciels et $(p-1)^2$ triadiques matricielles. Le coût total est donc :

$$T_N = p \times [\max_{i=1}^{2(p-1)} t_{comm}p^2 + \max_{i=1}^{(p-1)^2} t_{comm}((p-1)p^2) + \max_{i=1}^{2(p-1)}(c_{io}^p(3p^2) + t_{mm}^p)$$

$$+ \max_{i=1}^{(p-1)^2}(c_{io}^p(3p^2) + t_{ir}^p) + t_{rf}^p + t_{ri}^p] \quad (6.1)$$

Nous avons utilisé une version simplifiée de ce modèle pour estimer le coût de l'inversion de matrices de très grandes tailles par Gauss-Jordan sur différentes configurations réseau [PA04]. La complexité des trois opérations matricielles est approximativement égale à $2p^3$, on peut donc écrire l'équation (6.1) de la façon suivante :

$$T_N = p \times [2t_{comm}p^2 + 3t_{(2p^3 operations)}]$$

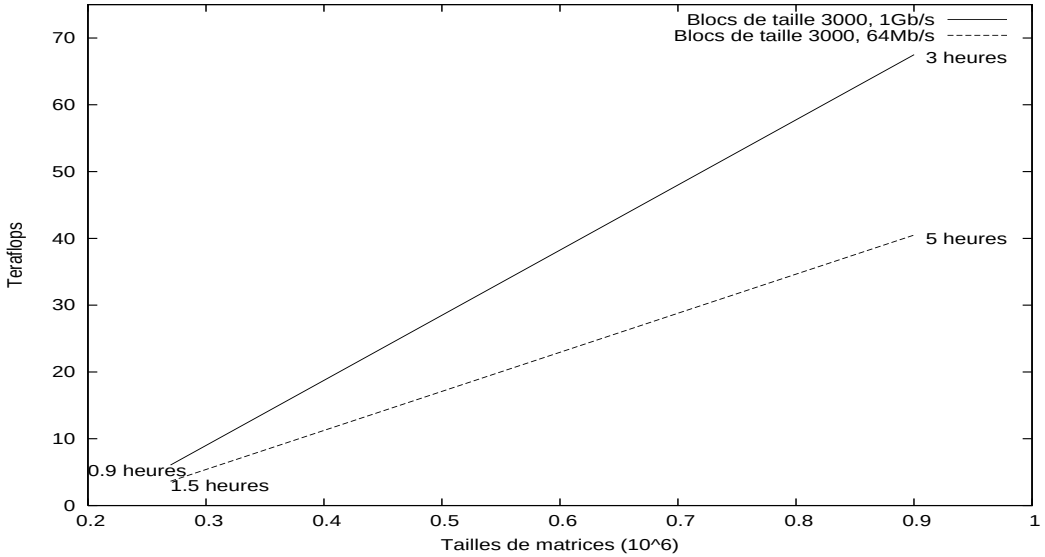
Nous évaluons cette formule sous les conditions suivantes :

- matrices denses de tailles 0.27×10^6 à 0.9×10^6,
- blocs de taille p égale à 3000,
- nombre de noeuds de calcul de 8100 (pour matrice de taille 0.27×10^6) à 90000 (pour matrice de taille 0.9×10^6),
- différents réseaux de communication cibles ; Internet classique à 56Kb/s, haut débit jusqu'à 1024Kb/s, très haut débit à 64Mb/s, et des liaisons de type VTHD et réseaux de recherche jusqu'à 1Gb/s.
- la puissance moyenne soutenue des noeuds de calcul pour l'algèbre linéaire de base est de 500 Mégaflops et la mémoire disponible allouable satisfait les conditions de distribution en sous-blocs énoncées auparavant et peut donc être variable.

Pour nos estimations, nous utilisons les résultats des tests de multicast décrits dans [HP]. Cette implantation utilise la bibliothèque MCL (MultiCast Library)[1] et le codage FEC, au dessus de MPICH-V2 et XtremWeb. La bibliothèque Zlib est aussi utilisé pour la compression de données. Cette implémentation atteint des taux de

[1] http ://www.inrialpes.fr/planete/people/roca/mcl/mcl.html

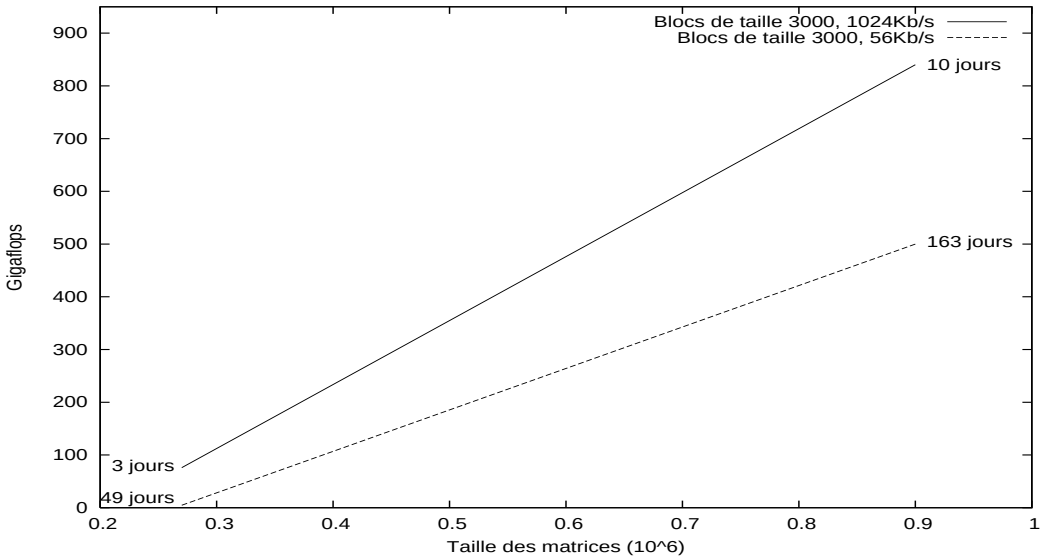
transmissions jusqu'à 13.6Mb/s en LAN Ethernet. Les graphes suivants montrent les résultats des estimations. On remarque que l'efficacité dépend de la vitesse des liens de communication. Elle peut atteindre 30% sur une plateforme de calcul de 90000 noeuds interconnectés par un réseau à 64Mb/s. Cependant, l'efficacité obtenue avec Internet bas débit est très basse (plus petite que 0.5%). Cela veut dire que la majeure partie du temps d'exécution est perdue en communications lentes. Par ailleurs, ces résultats montrent qu'il est possible d'inverser des matrices aussi larges que 0.9×10^6 en quelques heures et obtenir plusieurs téraflops avec des liens vraiment très haut débit.



6.5.3 Recouvrement des communications et du calcul

En plus des optimisations exploitées jusqu'ici en vue de réduire les communications, à savoir l'anticipation de migration et le clouage de données, on est aussi en mesure de recouvrir les communications par des calculs. En effet, du fait des dépendances dans le code, il y a toujours une limite au parallélisme, et donc à ces optimisations.

L'utilisation de la technique *out-of-core* dans ce contexte peut donc permettre le recouvrement des communications et calculs en implémentant des techniques de pipeline et en se basant sur des étages supplémentaires dans la décomposition et distribution des blocs, à l'image de ce qui a été fait plus haut dans ce chapitre. On sera donc en mesure de communiquer un sous-bloc résultat en faisant des calculs, et probablement des entrées/sorties sur d'autres. Ceci a pour effet de faire com-



muniquer au plus tôt une partie des données nécessaire au démarrage des calculs sur un autre noeud. Néanmoins, il faudra tenir compte de l'*overhead* associé aux communications et donc considérer une taille optimale des sous-blocs qui maximise le recouvrement, mais ceci reste difficile vu la forte hétérogénéité des architectures considérées. Ce schéma n'a pas été implémenté et est une perspective de ce travail.

Enfin, indépendamment de la technique *out-of-core* et des grilles informatiques, diverses études sur les possibilités de recouvrements dans les architectures et les logiciels ainsi que des études des gains que peuvent apporter ces techniques ont été présentées (essentiellement pour supercalculateurs à mémoire partagée ou distribuée) [Ram00]. À noter aussi que ces techniques font partie des optimisations classiques utilisées dans la compilation de langages data-parallèle ou de supports d'exécution.

6.6 Conclusion

L'émergence des grilles de calcul peut énormément augmenter le potentiel de résolution des problèmes scientifiques à grande échelle, seulement la conception et l'implémentation efficace de ces algorithmes reste problématique. Un des challenges est la définition d'un modèle de calcul général qui soit à la fois suffisamment détaillé pour refléter des aspects réalistes de ces environnements, mais aussi simple et assez abstrait et donc favorable à l'analyse.

Dans ce chapitre nous avons introduit un paradigme de calcul pour les applications matricielles sur la grille basé sur la programmation *out-of-core*, qui complète les techniques de placement de données introduites au chapitre précédent. Ce paradigme de calcul permet le passage à l'échelle et s'avère inévitable ; d'abord pour traiter les restrictions mémoire sur des machines de calcul non-dédiées, ensuite pour augmenter la taille maximale des applications, qui en pratique est limitée par les tailles mémoire. En effet, la croissance importante de la puissance des processeurs et des débits réseau définit les entrées/sorties comme le goulot d'étranglement le plus important dans le cadre des applications matricielles distribuées en grille.

La problématique de la programmation *out-of-core* n'est pas nouvelle. Nous proposons d'élargir son contexte d'utilisation aux grilles de calcul en utilisant une approche algorithmique qui nécessite la réécriture du code des tâches matricielles de base pour les applications visées. L'expérience menée dans cette étude sur des noyaux de l'algèbre linéaire, notamment les produits matrice-vecteur et matrice-matrice denses, ainsi que d'autres études sur le calcul *out-of-core* citées dans ce chapitre, montrent que le gain généré par l'effort algorithmique dépasse généralement les coûts arithmétiques éventuels. Par ailleurs, un algorithme conçu pour exploiter efficacement un niveau de mémoire particulier peut souvent bien exploiter les autres niveaux, en vu de l'exploitation éventuelle des caches ou registres. Nous avons aussi définit deux modélisations associées aux versions par blocs du produit matrice-vecteur et de l'inversion matricielle par Gauss-Jordan. Une évaluation de performance a été faite pour l'inversion par Gauss-Jordan sous ces hypothèses et sur différentes configurations réseau.

Au chapitre suivant, nous exposerons les défis qui demeurent pour l'algorithmique matricielle sur la grille ainsi que quelques perspectives que nous pouvons apporter à cette étude.

Chapitre 7

Du calcul global pour le calcul scientifique

Ce chapitre présente les challenges qui demeurent pour le calcul scientifique en environnements Grid, notamment autour des thèmes des paradigmes de calcul, évolution des environnements d'exécutions, et évaluation des performances. Nous faisons quelques remarques et propositions d'extension sur ces thématiques, et présentons quelques perspectives que nous pouvons apporter à nos travaux.

En premier lieu, nous présentons quelques propriétés requises pour le calcul scientifique sur la grille. Puis nous parlerons des environnements de calcul et aborderons enfin l'aspect 'performance' (en faisant parfois référence à l'environnement de grille utilisé dans cette étude, XtremWeb).

7.1 Problématique et contraintes

Nous avons évoqué, à plusieurs reprises dans ce manuscrit, les contraintes et problèmes dont doivent faire face les systèmes de calcul global, notamment, l'hétérogénéité des ressources, la dynamicité, la tolérance aux fautes, etc. Nous identifierons par la suite les caractéristiques souhaitables pour le calcul scientifique sur les grilles informatiques (liées ou pas au middleware utilisé).

Outre les problèmes liés à la gestion global de l'ensemble des ressources, notamment de placement et d'ordonnancement des tâches, la gestion de la localité et le placement des données sont des contraintes majeures du calcul scientifique sur la grille. Une observation globale du placement des données est nécessaire pour coordonner les calculs en fonction de l'endroit où se trouvent les données. Ceci nécessite le clouage des données et l'anticipation des transferts. Certains systèmes de calcul global proposent de maintenir des références sur les tâches ou données des applications parallèles. L'environnement XtremWeb utilisé dans nos applications ne propose

pas ce mécanisme. Aussi, vu le caractère plus ou moins volatile des ressources, il faudrait mettre en place des solutions de réplication et de cohérence des données. De plus, l'exigence en terme de performance peut amener à introduire des mécanismes de cache.

Les techniques d'anticipation de migration (ou pré-déploiement) sont aussi réalisables dans plusieurs algorithmes de calcul matriciel. En réalité, le taux d'anticipation est variable et est fonction du choix de la taille des données, et donc des tâches et de leur ordonnancement. Ceci est un problème d'optimisation dépendant de l'application (distribution des données notamment) et des temps de calcul, qui doivent recouvrir cette anticipation. Cela nécessite un ordonnancement efficace et des solutions d'indexation de ressources basés spécialement sur un critère de présence de telles ou telles données sur les noeuds de calcul. Il faut aussi disposer de systèmes de mise à jour de ces informations qui doivent rester efficaces lors de changement d'emplacements, comme par exemple lors d'une re-distribution de données pour un ré-équilibrage de charge ou après une défaillance.

Les techniques d'exploitation de la localité des données permettent de traiter la problématique liée à la granularité des problèmes d'algèbre linéaire. En effet, le calcul matriciel présente souvent des tâches parallèles avec grain de calcul fin (c'est-à-dire des tâches concourantes ou interdépendantes). Cette classe de calcul est très exigeante pour le système et un cas délicat pour les grilles. Les techniques de placement de données permettent de résoudre ces problèmes dans les meilleures conditions possibles. Disposer de tels mécanismes intégrés aux middlewares de grille serait un atout considérable pour le calcul scientifique.

Dans notre contexte, la problématique de la gestion globale est aussi plus difficile car la plateforme est partagée par plusieurs applications ou plusieurs utilisateurs. Les ressources sont donc non-dédiées. Le système devrait considérer ce partage de la plateforme et disposer de mécanismes d'ordonnancement efficaces. Une priorité devient aussi d'adapter la conception algorithmique et donc de s'accommoder en amont à la quantité de ressources disponible. La technique de programmation *out-of-core* proposée ici est une solution en ce qui concerne les contraintes mémoire sur la plateforme de calcul. Une perspective de ce travail est l'utilisation du recouvrement des communications, des calculs et des entrées/sorties en implémentant des techniques de pipeline. Pour les applications testées (le produit matrice-vecteur et l'inversion matricielle par Gauss-Jordan) on peut se baser sur des étages supplémentaires dans la décomposition/distribution en blocs pour communiquer au plus tôt une partie des données nécessaire au démarrage des calculs sur d'autres noeuds.

Les technologies d'agents [CY][FTSK][XK][TKV$^+$] semblent aussi être d'un grand intérêt pour les grilles. En effet, les ordonnanceurs et applications doivent être adaptables aux variations de l'environnement, si possible de manière dynamique. Dans

ce cadre, la mobilité des agents peut rendre le système capable de s'accommoder dynamiquement aux conditions d'exécution. Chaque agent peut avoir une mission différente ; de surveillance du réseau ou de notification et gestion d'événements. Il existe plusieurs implémentations de middleware fournissant des mécanismes pour la programmation à base d'agents telles que JavAct, JADE, AJANTA, etc. Pour les environnements et applications sur les grilles de calcul, ces systèmes peuvent servir de base à l'implémentation de modèles, protocoles ou stratégies de communication, d'ordonnancement, d'adaptation de schéma de calcul, de migration de données, d'équilibrage de charge, etc.

7.2 Environnements d'exécution et de programmation

D'une manière générale, les systèmes de calcul global actuels considèrent les applications parallèles de type multi-paramètres ou maître-esclave. Cela se traduit généralement par une gestion statique des rôles, en l'occurrence pour XtremWeb un unique coordinateur qui centralise l'exécution. Ceci est inapproprié pour le calcul scientifique. Un schéma décentralisé, ou hiérarchique (à la JXTA) est requis, notamment pour le passage à l'échelle et pour éviter les goulots d'étranglement. La possibilité d'avoir un serveur par site dans nos plateformes de calcul aurait été préférable. Aussi, la gestion des données et fichiers, et la tolérance aux fautes, doit être cachée au programmeur, ce qui s'est avéré ne pas être tout à fait le cas dans XtremWeb. À noter tout de même le bien-fondé de l'utilisation de la compression de données dans ce système.

En effet, sur des tests de compression sur 28 matrices de différentes tailles (de 4Mo à 100Mo) issues de *matrix market*[1], nous avons obtenu des taux de compression de 74.6% (pour valeurs codées en 64bits) à 84.4% (pour valeurs codées en 32bits) en utilisant deux formats de compression (gzip et bzip2). Les temps de compression/décompression seront naturellement variables sur les noeuds d'une grille, néanmoins, des études utilisant la compression de données en environnements parallèles comme dans [HP] (un multicast utilisant le protocole ALC, le codage FEC et la bibliothèque Zlib), ou la version de MPI utilisant une compression de messages à la volé appelée cMPI [KBS], montrent des gains en temps de calcul très significatifs atteignant les 98%. La bibliothèque AdOC[2] fournit un ensemble de fonctions permettant de transmettre les fichiers sur le réseau en utilisant la compression dynamique adaptée à l'environnement. Elle a été intégré a NetSolve et a amélioré les performances d'un facteur de 5 sur des liens de types Internet, mais pas de différence sur des réseaux à haut débit. La compression est donc nécessaire car elle améliore les résultats dans tous les cas étudiés.

[1] Au format Harwell-Boeing. http ://math.nist.gov/MatrixMarket/
[2] http ://sedre.loria.fr/proposition/cd/cdrom/prog/unix/adoc/fra.htm

En outre, la programmation des applications matricielles avec l'API Java d'XtremWeb n'était pas triviale, car elle ne fournit pas de fonctionnalités de gestion des dépendances et synchronisations, ni de réutilisation ou de gestion de localité des données et/ou résultats. L'API d'OmniRPC est plus bas niveau et fournit quelques fonctionnalités intéressantes mais reste limitée notamment pour les calculs interdépendants et les tailles des fichiers de données. Le manque de *frameworks* génériques de haut niveau et d'outils d'aide à la programmation pour le calcul scientifique est évident pour les grilles. Les interfaces ou langages pour les Web et Grid services basés sur XML et les *workflows* (tels que WSFL, XLang, GSFL...), et les langages de *workflow* (tels que DAGMan de Condor, YvetteML, Triana...) constituent une approche haut niveau naturelle et souhaitable pour les applications matricielles.

Les *workflows* fournissent donc les fonctionnalités de gestion de dépendances et placement des tâches et données nécessaires aux applications scientifiques et sont bien adaptés aux environnements de calcul global. Néanmoins, l'interopérabilité, ou interfaçage, des langages de *workflow* avec les plateformes de grille existants est limitée. Il existe aussi plusieurs modèles de représentation et composition de *workflows*. Nous mentionnons celui utilisé par YML [DP] basé sur un modèle de composants qui permet une plus grande indépendance avec les middlewares sous-jacents (actuellement limité à XtremWeb) et facilite le partage, la réutilisation et la sauvegarde d'applications.

7.3 Evaluation et reproduction de performances

La mesure de la performance est une question très difficile pour les environnements de calcul à grande échelle. En effet, il existe un nombre considérable de paramètres dans ces systèmes : l'organisation et l'architecture de la plateforme, son implémentation (langages utilisés), les protocoles sous-jacents (notamment de communication), et les contraintes liées aux ressources (hétérogénéité, dynamicité, etc). Il existe donc une réelle difficulté expérimentale, d'abord à identifier les paramètres discriminants significatifs, ensuite à mesurer et quantifier leur impact et aussi à reproduire les performances des applications parallèles, et donc des incertitudes au niveau de la conception des applications et de l'évaluation algorithmique.

Il existe très peu d'outils de prédiction de performances pour applications parallèles en environnements distribués. Dimemas [BEG+a] du projet DAMIEN est un outil de prédiction pour applications à passage de messages sur des architectures distribuées hétérogènes. Il prend en entrée une trace de l'exécution de l'application cible avec des captures de l'information CPU, du modèle de communication (point à point ou collective) et un fichier de configuration qui contient un ensemble de paramètres modélisant l'architecture cible. Un ensemble de mesures (données dans [BEG+b]) sur des applications à passage de messages (utilisant PACX-MPI) donne parfois un facteur de 4 entre les valeurs réelles et les prédictions (75% d'erreur), sur

un nombre de processeurs pourtant très restreint, en l'occurrence de 4 à 14 noeuds de calcul. Performance Prophet [PF] est un autre outil de modélisation et prédiction pour applications parallèles et distribuées. Le modèle de performance se base sur une description UML des applications (en utilisant l'outil de modélisation Teuta). L'estimateur de performance est un ensemble de classes C++ qui modélise les fonctions de base et les composants de l'architecture (le modèle UML est transformé en une représentation C++ pour l'évaluation de performances). Les tests dans [PF] sur des clusters de quadri-processeurs (de 8 à 16 noeuds) pour une application MPI/Fortran a donnée des erreurs de 2 à 24%. On peut aussi citer le système PACE [NKP+00] ou les outils de simulations décrits au chapitre 3 (tel que SimGrid), qui sont plutôt dédiés aux stratégies d'ordonnancement qu'à la conception d'applications distribuées.

Ces systèmes sont limités à un modèle de programmation, un middleware ou une plateforme donnée et ne considèrent pas l'extensibilité du système. Toutefois, une généralisation est tout bonnement impossible car chaque plateforme a sa propre organisation et spécificités des ressources. Aussi, chaque middleware a son architecture, son implémentation, ses protocoles, ses outils et langages de développement, ses paradigmes de calcul, etc. Tous ces paramètres influent sur les performances. Dans nos expériences, il était fréquent d'avoir des différences de performances, parfois considérables, sur des exécutions à priori sous les mêmes conditions (et même avec les noeuds de calcul déployés et dédiés de Grid'5000 par exemple). Il nous était généralement impossible d'expliquer cela. La solution serait de tracer chacune des exécutions afin d'avoir l'état général de la plateforme tout au long des tests. Il faudra ensuite être capable d'analyser et traiter ces données via des outils appropriés, notamment de visualisation. Il n'y a pas, à notre connaissance, un environnement de grille de calcul qui propose de tels outils système.

Néanmoins, au niveau de la conception algorithmique, les outils d'émulation et grilles expérimentales, tels que Grid eXplorer et Grid'5000, peuvent aider à l'étude de l'impact de certains aspects des grilles sur les noyaux de l'algèbre linéaire car ils permettent, dans une certaine mesure, de reproduire les conditions expérimentales. Les problématiques relevant des grilles de calcul seront donc relativement maîtrisées et les résultats généralisables. Par ailleurs, ces systèmes ne présentent pas de problèmes de validation de résultats inhérents aux simulateurs.

7.4 Conclusion

L'utilité du partage des ressources pour le calcul global et pair-à-pair n'est plus à démontrer. Les puissances de calcul phénoménales atteintes avec les systèmes pionniers attestent du potentiel considérable exploitable. L'omniprésence actuelle (et très probablement future) des grappes de calcul haute performance (plus de

60% des configurations du Top500[3]) et le nombre de ressources sans cesse croissant connectées à Internet (qui atteindra le milliard en 2015) justifient pleinement cet intérêt. Néanmoins, les modèles de calcul actuels limitent fortement le champ d'application des grilles, toutefois nous avons montré dans cette étude que beaucoup d'aspects des applications matricielles peuvent être exploités pour fournir des niveaux de performance beaucoup plus élevés sur ces environnements.

Nous avons évoqué dans ce chapitre les défis actuels du calcul matriciel et plus largement du calcul scientifique sur la grille et fait quelques propositions sur les thèmes liés à notre contexte d'étude. La mise en place de schéma optimal des flux de données (des mémoire locales jusqu'au réseau) reste, à notre avis, l'aspect le plus important pour des implémentations efficaces sur la grille. Au delà de cette constatation, le champ de recherche ouvert est très vaste pour l'algorithmique sur la grille, et des outils proposant les abstractions requises au niveau système et au niveau utilisateur sont nécessaires et sont une suite logique à l'effort au niveau application fournit dans cette étude.

[3]Sur le classement établi en juin 2005.

Chapitre 8

Conclusion

Les applications matricielles ont des caractéristiques naturellement parallèles et de calcul intensif et sont donc candidates aux grilles informatiques. Néanmoins, un travail d'adaptation d'algorithmes existants ou de conception d'algorithmes adaptés est nécessaire. L'algorithmique matricielle parallèle doit contribuer à l'affranchissement vis-à-vis des limites imposées par la nature même des grilles de calcul, notamment les capacités matérielles des architectures et machines cibles.

Dans cette étude, nous avons analysé les caractéristiques inhérentes à certaines applications matricielles en terme de placement et distribution des données et exigences mémoire, et nous avons proposé des techniques de stockage persistant ; anticipation de migration et clouage de données, et de programmation *out-of-core*. Ces techniques ont été utilisées pour mettre en place des applications matricielles sur des grilles de calcul réparties sur deux continents. Des noyaux de l'algèbre linéaire ; le produit matrice-vecteur, et l'inversion matricielle par Gauss-Jordan ont été exécutés sous différentes conditions et configurations, et les résultat montrent l'efficacité de ces techniques pour améliorer les performances du calcul matriciel sur la grille.

Le travail réalisé vise à l'étude de la possibilité d'exploiter des puissances de calcul en local et sur réseaux étendus pour exécuter des applications matricielles, à priori dédiées aux supercalculateurs et architectures fortement couplées, physiques ou simulées. Ce type de développement est un problème très vaste. Le nombre considérable de paramètres concernés, étudié tout au long de cette thèse, rend toute synthèse à ce sujet très difficile. Néanmoins, les nombreux exemples d'algorithmes et de configurations étudiés nous ont permis de conclure qu'il est indispensable de gérer explicitement le placement des données tout au long du processus de parallélisation sur les grilles de calcul non-dédiées.

Ceci ne doit pas être un effort exclusivement algorithmique. En effet, les contraintes liées au calcul global, notamment d'hétérogénéité et de dynamicité, peuvent être concidérées à d'autres niveaux ; système ou utilisateur, en proposant les abstractions

requises dans les outils logiciels et environnements de gestion et de développement sur les grilles. À travers notre approche expérimentale, nous avons mis en évidence les préférences de développement en terme de performance. Ceci est toutefois lié au middleware d'expérimentation, en l'occurrence XtremWeb, mais le manque de fonctionnalités avancées sur ce système réduit le nombre de paramètres à considérer (et les *overheads* générés), qui peuvent se résumer à l'architecture, le langage de programmation ou les protocoles sous-jacents.

Le modèle proposé est assez nouveau, se basant sur le stockage persistant et la programmation *out-of-core*, avec des distribution par blocs des applications matricielles visées. Il nécessite une gestion explicite des flux de données à tous les niveaux ; d'abord en choisisant un placement efficace pour les blocs des matrices, en utilisant le clouage et l'anticipation d'envoi, pour réduire les besoins en communications et augmenter le rapport calcul/communication, ensuite une gestion algorithmique des entrées/sorties disque afin d'éviter toute pagination. Ce modèle offre un niveau de scalabilité inatteignable par une approche maître-esclave classique et est largement plus performant. Les tâches matricielles utilisées sur les noeuds sont de plus grandes tailles et les exécutions utilisent moins de ressources. Cette taille est naturellement adaptable en fonction de la mémoire allouable disponible. Par ailleurs, nous avons utilisé pour les tâches matricielles de base les routines de bibliothèques d'algèbre linéaire largement utilisées et qui offrent de bonnes performances ; les BLAS et LAPACK.

D'un autre côté, les études comparatives réalisées, entre les différents cas d'exécution et avec des implémentations classiques à passage de message utilisant la bibliothèque MPICH, nous ont permis d'établir un ordre de préférence, qui reste néanmoins relatif car lié à une application donnée, en l'occurrence la version par bloc de l'inversion matricielle par Gauss-Jordan. La gestion mémoire est alors préférée en terme de gain de performances par rapport aux gains liés aux communications, mais nous avons remarqué qu'il est toujours plus intéressant de les associer. À noter que la programmation *out-of-core* est effectuée au niveau local et précède la parallélisation proprement dite. En revanche, les tests sur Grid'5000 ont montré que la gestion *out-of-core* n'est pas indispensable dans le cas de plateformes de calcul dédiées avec assez de mémoire sur les noeuds de calcul pour les tâches considérés. L'*overhead* associé à cette gestion est plus important qu'un gain éventuel par rapport à une pagination initiale qui est en l'occurrence inexistante.

Par ailleurs, nous avons mentionné que le déploiement, l'administration et les tests sous ces outils expérimentaux impliquent un travail de mise au point et adaptation continu car ces outils évoluent fréquemment, et malgré le soin certain qui est apporté à ces réalisations, le travail de débogage demeure important. Nous avons donc cité les limites de l'interface de développement du middleware, notamment en terme de modèle de calcul et fonctionnalités de gestion de dépendances et placement

des tâches et données. Son architecture centralisée et les restrictions liées à son API, pour les tailles de fichiers de données et les limites de la gestion mémoire de Java (lié au *Garbage Collector*) ne permettent pas le passage à l'échelle pour le calcul matriciel. Ainsi, atteindre un bon niveau de scalabilité et de performance sur la grille reste un défi important. Les contraintes et les limitations sur des infrastructures existantes et les outils sous-jacents, comme celles décrites ici, rendent le chemin à la scalabilité plus difficile. Les niveaux supplémentaires de distributions en sous-blocs pour des implémentations *out-of-core* peuvent contribuer à régler ces limites de tailles, qui peuvent avoir d'autres causes, liée au systèmes de fichiers par exemple pour de très larges problèmes. Ces étages supplémentaires dans la décomposition et distribution peut aussi aider à implémenter des techniques de recouvrement des communications et calculs.

Enfin, nous avons évoqué les choix requis en terme d'architectures, d'outils d'implémentation sous-jacents et d'interopérabilité avec des *framework* de haut niveau pour le calcul matriciel sur les grilles et l'aide à la programmation pour l'utilisateur final, pour conclure que les champs de recherche liés à ces problématiques reste très large.

Annexe A : Extrait de code client

Ce code client Java est un extrait du programme de l'inversion matricielle par Gauss-Jordan par blocs (sans la gestion des re-soumissions pour plus de lisibilité). Il donne un aperçu sur l'envoi de tâches, en l'occurrence la distribution des produits de blocs pour les boucles (1) et (2) de l'algorithme donné en section 5.2.2.

```
/***
*** Dans la boucle d'indice 'step' (equivalent a 'k').
***/

...
// Recuperer le resultat de l'inversion de l'etape 'step'
 System.out.println(" Getting result for inversion .."+step) ;
 boolean[] available = new boolean[nodes1] ;
 for (int i=0 ; i<nodes1 ; i++)
   available[i] = false ;
 int total = 0 ;
 while ( total < nodes1 ) {
  for (int k=0 ;k<nodes1 ; k++) {
   if (available[k] == false) {
    try {
    if (XWGaussJordan.comm.jobStatus(mw[k]) == WorkStatus.COMPLETED) {
       System.out.println(" Job "+mw[k].getUID()+" completed..") ;
       XWGaussJordan.comm.getJobResult(mw[k].getUID() ;)
       XWGaussJordan.tcp.getResult(mw[k].getUID(),
                                          mw[k].getUID()+"_out.zip") ;
       Debug.Info("getting result " + mw[k].getUID()) ;
       processFile(mw[k].getUID()+"_out.zip", k) ;
       total++ ;
       available[k] = true ;
       IDs.add(mw[k].getUID()) ;

       //dezipper le fichier resultat pour les operations suivante
       zipper.setFileName(mw[k].getUID()+"_out.zip") ;
```

```
        zipper.unzip("./");
      }

  // Re-soumission en cas d'erreur
      if (XWGaussJordan.comm.jobStatus(mw[k]) == WorkStatus.ERROR) {
        XWGaussJordan.comm.deleteJobs(mw[k].getUID());
        mw[k] = new MobileWork("Inversion_"+k+step);
        mw[k].setServer (XWGaussJordan.config.getCurrentServer ());
        mw[k].setServer (XWGaussJordan.config.getCurrentServer ());
        mw[k].setApplicationName ("Inverse");
        mw[k].setCmdLine (cmdLine);

      //tests sur re-soumissions
      if (XWGaussJordan.comm.submitJob(mw[k], session.getName(),
                                          group.getName()) == null) {
        System.out.println(" Can't submit "+mw[k].getUID());
        Debug.Info(" job "+mw[k].getUID() + " : can't submit");
      } else {
        System.out.println(" Job "+mw[k].getUID()+" re-submitted ");
        Debug.Info(" submitted "+mw[k].getUID());
      }
      }
     } catch (Exception exp) {
     Debug.Warning("exception in getting result : "+ exp);
     }
    }
   }
}

/***
*** Envoi des produit de blocs Aki=Bkk*Aki
***/

int no_job=0;
for(int l=step+1; l<taille; l++){

   //creation des fichiers zip
   String[] inproduct1 = new String[]{"A_"+step+l,"B_"+step+step};
   zipper.setFileName("A_"+step+l+".zip");
   zipper.zip(inproduct1);

   //soumission des taches
   String cmdLine = "A_"+step+l+" "+"B_"+step+step+" "+"A_"+step+l;
```

```
    mw2[no_job] = new MobileWork("MProduct_"+no_job+step) ;
    mw2[no_job].setServer (XWGaussJordan.config.getCurrentServer ()) ;
    mw2[no_job].setApplicationName ("MProduct") ;
    mw2[no_job].setCmdLine (cmdLine) ;
    mw2[no_job].setDirin("./", "A_"+step+l+".zip") ;

    //tests sur soumissions
    if (XWGaussJordan.comm.submitJob(mw2[no_job], session.getName(),
                                     group.getName()) == null) {
      System.out.println(" Can't submit "+mw2[no_job].getUID()) ;
      Debug.Info(" job "+mw2[no_job].getUID() + " : can't submit") ;
    } else {
      System.out.println(" Job "+mw2[no_job].getUID()+" submitted ") ;
      Debug.Info(" submitted "+mw2[no_job].getUID()) ;
    }
    no_job++ ;
  }

  for(int l=0 ; l<taille ; l++){

    /***
    *** Envoi des produits de blocs Bik=-Aik*Bkk
    ***/

    if(l !=step){

      //creation de fichiers zip
      String[] inproduct2 = new String[]{"A_"+l+step,"B_"+step+step} ;
      zipper.setFileName("B_"+l+step+".zip") ;
      zipper.zip(inproduct2) ;

      //soumission des taches
      String cmdLine = "A_"+l+step+" "+"B_"+step+step+" "+"B_"+l+step ;
      mw2[no_job] = new MobileWork("MProduct_"+no_job+step) ;
      mw2[no_job].setServer (XWGaussJordan.config.getCurrentServer ()) ;
      mw2[no_job].setApplicationName ("MProduct_1") ;
      mw2[no_job].setCmdLine (cmdLine) ;
      mw2[no_job].setDirin("./", "B_"+l+step+".zip") ;

      //tests sur soumissions
      if (XWGaussJordan.comm.submitJob(mw2[no_job], session.getName(),
                                       group.getName()) == null) {
        System.out.println(" Can't submit "+mw2[no_job].getUID()) ;
```

```
    Debug.Info(" job "+mw2[no_job].getUID() + " : can't submit");
   } else {
    System.out.println(" Job "+mw2[no_job].getUID()+" submitted ");
    Debug.Info(" submitted "+mw2[no_job].getUID());
   }
   no_job++;

/***
*** Envoi des produits de blocs Bki=Bkk*Bki
***/

if (l<step) {

   //creation de fichiers zip
   String[] inproduct_2 = new String[]{"B_"+step+l,"B_"+step+step};
   zipper.setFileName("B_"+step+l+".zip");
   zipper.zip(inproduct_2);

   //soumission des taches
   String cmdLine = "B_"+step+l+" "+"B_"+step+step+" "+"B_"+step+l;
   mw2[no_job] = new MobileWork("MProduct_"+no_job+step);
   mw2[no_job].setServer (XWGaussJordan.config.getCurrentServer ());
   mw2[no_job].setApplicationName ("MProduct");
   mw2[no_job].setCmdLine (cmdLine);
   mw2[no_job].setDirin("./", "B_"+step+l+".zip");

   //tests sur soumission
   if (XWGaussJordan.comm.submitJob(mw2[no_job], session.getName(),
                                    group.getName()) == null) {
    System.out.println(" Can't submit "+mw2[no_job].getUID());
    Debug.Info(" job "+mw2[no_job].getUID() + " : can't submit");
   } else {
    System.out.println(" Job "+mw2[no_job].getUID()+" submitted ");
    Debug.Info(" submitted "+mw2[no_job].getUID());
   }
    no_job++;
}

/***
*** Recuperer les resultats des produits pour les triadiques
***/

System.out.println(" Getting results for Product 1 ...");
```

```
boolean[] available2 = new boolean[nodes2];
for (int i=0; i<nodes2; i++)
 available2[i] = false;
 int total1 = 0;
 while ( total1 < nodes2 ) {
  for (int k2=0;k2<nodes2; k2++){
   if (available2[k2] == false) {
    if (XWGaussJordan.comm.jobStatus(mw2[k2]) == WorkStatus.COMPLETED) {
     try {
      System.out.println(" Job "+mw2[k2].getUID()+" completed..");
      XWGaussJordan.comm.getJobResult(mw2[k2].getUID());
      XWGaussJordan.tcp.getResult(mw2[k2].getUID(),
                                          mw2[k2].getUID()+"_out.zip");
      Debug.Info("getting result " + mw2[k2].getUID());
      processFile(mw2[k2].getUID()+"_out.zip", k2);
      total1++;
      available2[k2] = true;
      IDs.add(mw2[k2].getUID());

      //dezipper pour l'etape suivante
      zipper.setFileName(mw2[k2].getUID()+"_out.zip");
      zipper.unzip("./");
     } catch (Exception exp) {
      Debug.Warning("exception in Launcher : "+ exp);
      System.out.println("exception in getting results : "+ exp);
     }
    }
   }
  }
 }
...
```

Bibliographie

[AAB+00] Y. Amir, B. Awerbuch, A. Barak, S. Borgstrom, and A. Keren. An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster. In USA IEEE CS Press, editor, *IEEE Transactions on Parallel and Distributed Systems*, volume 11, pages 760–768, July 2000.

[AAW+02] B. A. Allen, R. C. Armstrong, A. P. Wolfe, J. Ray, D. E. Bernholdt, and J. A. Kohl. The CCA Core Specification in a Distributed Memory SPMD Framework. *Concurrency and Computation : Practice and Experience*, 14(5) :323–345, 2002.

[ABB+] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics.

[ABE+] P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, and Y. J. Wu. PLAPACK : Parallel Linear Algebra Package. In Proceedings of the SIAM Parallel Processing Conference, 1997.

[ACK+] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werhimer. SETI@home : An Experiment in Public-Ressource Computing. Communications of the ACM, Nov. 2002, Vol. 45 No. 11, pp. 56-61.

[AISS] A. Alexandrov, M. Ibel, K. Schauser, and C. Scheiman. SuperWeb : Towards a Global Web-Based Parallel Computing Infrastructure. Proceedings of the 11th International Parallel Processing Symposium (IPPS'97), Geneva, 1997.

[And] D. P. Anderson. BOINC : A System for Public-Ressource Computing and Storage. GRID 2004, fifth IEEE/ACM international workshop on Grid Computing. SuperComputing 2004, Pittsburgh.

[Aou02] L. M. Aouad. Méthodes hybrides, approche orientée objet et parallélisme. Rapport de DEA, Université de Paris-XI, Orsay, 2002.

[AP] L. M. Aouad and S. Petiton. Experimentations and Programming Paradigms for Matrix Computing on Peer to Peer Grid. GRID 2004, fifth IEEE/ACM international workshop on Grid Computing. SuperComputing 2004, Pittsburgh.

[APS] L. M. Aouad, S. Petiton, and M. Sato. Grid and Cluster Matrix Computation with Persistent Storage and Out-of-Core Programming. The 2005 IEEE International Conference on Cluster Computing, September 2005. Boston, Massachusetts.

[AS01] R. Alfieri and F. Spataro. MPICH-G2 performance evaluation on PC clusters. Technical report, Dipartimento di Fisica, Universita Degli Studi Di Parma, Italia., 2001.

[BAG] R. Buyya, D. Abramson, and J. Giddy. An Economy Driven Resource Management Architecture for Global Computational Power Grids. The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, USA, June 26-29, 2000.

[BCC+03] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4), 2003.

[BCD] R. Buyya, S. Chapin, and D. DiNucci. Architectural Models for Resource Management in the Grid. The First IEEE/ACM International Workshop on Grid Computing, GRID 2000. December 17-20, 2000. Bangalore, India.

[BCH+] A. Bouteiller, F. Cappello, T. Hérault, G. Krawezik, P. Lemarinier, and F. Magniette. MPICH-V2 : a Fault Tolerant MPI for Volatile Nodes based on the Pessimistic Sender Based Message Logging. Proceedings of The IEEE/ACM SuperComputing 2003 Conference. Phoenix USA, November 2003.

[BCL+] J. P. Baud, J. Casey, S. Lemaitre, C. Nicholson, D. Smith, and G. Stewart. LCG Data Management : From EDG to EGEE. UK e-Science All Hands Meeting, 2005.

[BCR96] R. Bordawekar, A. Choudhary, and J. Ramanujam. Compilation and Communication Strategies for Out-of-core programs on Distributed Memory Machines. volume 38, pages 277–288, 1996.

[BDV94] G. Burns, R. Daoud, and J. Vaigl. LAM : An Open Cluster Environment for MPI. In *In Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[BEG+a] R. M. Badia, F. Escalé, E. Gabriel, J. Giménez, R. Keller, J. Labarta, Matthias, and S. Muller. DIMEMAS : Predicting MPI applications behavior in Grid environments. Workshop on Grid Applications and Programming Tools (GGF8), 2003.

[BEG+b] R. M. Badia, F. Escalé, E. Gabriel, J. Giménez, R. Keller, J. Labarta, and M. S. Muller. Performance Prediction in a Grid Environment. 1st European AcrossGrid conference. Santiago de Compostela, 2003.

[BEG+97] S. Balay, V. Eijkhout, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202, 1997.

[BH] C. Brian and G.-M. Hector. Peer-to-Peer Resource Trading in a Reliable Distributed System. First International Workshop on Peer-To-Peer Systems, 2002.

[BKKW96] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wyckoff. Charlotte : Metacomputing on the web. In *Proceeding of the 9th International Conference on Parallel and Distributed Computing Systems (PDCS'96)*, 1996.

[bla] Basic linear algebra subprograms. a quick reference guide for the blas. University of Tennessee. Oak Ridge National Laboratory.

[Bor00] S. Borgstrom. *A Cost-Benefit Approach to Resource Allocation in Scalable Metacomputers*. PhD thesis, The Johns Hopkins University, Baltimore, Maryland, 2000.

[BPSa] J-M. Busca, F. Picconi, and P. Sens. Pastis : a Highly-Scalable Multi-User Peer-to-Peer File System. EuroPar 2005, Lisboa, Portugal, September 2005. LNCS.

[BPSb] J-M. Busca, F. Picconi, and P. Sens. Pastis : un système de fichiers pair è pair multi-écrivain passant l'échelle. In DistRibUtIon de Données à grande Echelle 2004 (DRUIDE 04). Domaine du Port-aux-Rocs, Le Croisic, France.

[BSC+] R. Batchu, A. Skjellum, Z. Cui, M. Beddhu, J. P. Neelamegam, Y. Dandass, and M. Apte. MPI/FT : Architecture and Taxonomies for Fault-Tolerant, Message-Passing Middleware for Performance-Portable Parallel Computing. Proceeding of the first International Symposium on Cluster Computing and the Grid, 2001.

[Buy02] R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, School of Computer Science and Software Engineering Monash University, Melbourne, Australia, 2002.

[BWC+03] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, S. Spring, A. Su, and D. Zagorodnov. Adaptive Computing on the Grid using AppLeS. *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, 14(4) :369–382, 2003.

[Car00] E. Caron. *Calcul numérique sur données de grande taille*. Thèse de Doctorat, Université de Picardie Jules Verne, 2000.

[Casa] H. Casanova. Distributed Computing Research Issues for Grid Computing. Quarterly Newsletter for the ACM Special Interest Group on

Algorithms and Computation Theory (SIGACT News), Vol. 33, Num. 2, September 2002.

[Casb] H. Casanova. Simgrid : A Toolkit for the Simulation of Application Scheduling. Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, 2001, Brisbane, Australia.

[CBOW] H. Casanova, F. Berman, G. Obertelli, and R. Wolski. The AppLeS Parameter Sweep Template : User-Level Meddleware for the Grid. SuperComputing 2000. Dallas, USA.

[CCEB03] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Eutropia : Architecture and Performance of an Entrprise Desktop Grid System. *Journal of Parallel and Distributed Computing*, 63(5) :597–610, 2003.

[CCI+97] P. Cappello, B. Christiansen, M. Ionescu, M. Neary, K. Schauser, and D. Wu. Javelin : Internet-Based Parallel Computing Using Java. *Concurrency : Practice and Experience*, 1997.

[CD05] Eddy Caron and Frédéric Desprez. DIET : A Scalable Toolbox to Build Network Enabled Servers on the Grid. Rapport de recherche, Laboratoire de l'Informatique du Parallélisme (LIP), 2005.

[CDL+05] E. Caron, F. Desprez, J.-Y. L'Excellent, C. Hamerling, M. Pantel, and C. Puglisi-Amestoy. Use of A Network Enabled Server System for a Sparse Linear Algebra Application. Rapport de recherche, Laboratoire de l'Informatique du Parallélisme, École Normale Supérieur de Lyon, 2005.

[Cer04] H. Cervantes. *Vers un modèle à composants orienté services pour supporter la disponibilité dynamique.* Thèse de Doctorat, Université de Grenoble 1, 2004.

[CFF+04a] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From Open Grid Services Infrastructure to WS Resource Framework : Refactoring and Evolution. The Globus Alliance, 2004.

[CFF+04b] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework. The Globus Alliance, 2004.

[CGJ+00] B. Carpenter, V. Getov, G. Judd, A. Skjellum, and G. Fox. MPJ : MPI-like Message Passing for Java. *Concurrency : Practice and Experience*, 12(11) :1019–1038, 2000.

[CJ] Y. Caniou and E. Jeannot. Ordonnancement pour la grille : une extension de MCT. RENPAR'14, 2002.

[Coz03] O. Cozette. *Contribution systèmes pour le traitement de grandes masses de données sur grappes.* Thèse de Doctorat, Université de Picardie Jules Verne, 2003.

[CRLS03] N. Coleman, R. Raman, M. Livny, and M. Solomon. Distributed Policy Management and Comprehension with Classified Advertisements. Technical report, University of Wisconsin - Madison Computer Sciences Department, 2003.

[Cro03] C. S. Crow. DistSolve : An Open Source Web Service for Solving Sparce Systems of Equantions. *Hopkins Undergraduate Reseach Journal*, 2003.

[CU] E. Caron and G. Utard. Parallel Out-of-Core Matrix Inversion. IPDPS'02. The 16th International Parallel and Distributed Processing Symposium, 2002.

[CY] R. Yu Chen and B. Yeager. Java Mobile Agents on Project JXTA Peer to Peer Platform. Proceedings of the 36th Hawaii International Conference on System Sciences, HICSS03.

[Dan03] J. Daniel. *Services Web - Concepts, techniques et outils*. Vuibert, 2003.

[DD00] E. D'Azevedo and J. Dongarra. The Design and Implementation of the Parallel Out-of-core ScaLAPACK LU, QR and Cholesky Factorisation Routines. *Concurrency : Practice and Experience*, 12(15), 2000.

[Den03] A. Denis. *Contribution à la conception d'une plateforme haute performance d'intégration d'exécutifs communicants pour la programmation des grilles de calcul*. Thèse de Doctorat, Université de Renne-I, 2003.

[DF] C. Dumitrescu and I. Foster. GangSim : A Simulator for Grid Scheduling Studies. Proceeding of the IEEE International Symposium on Cluster Computing and the Grid, CCGrid'05. UK, 2005.

[DFP+96] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-Way : Wide Area Visual Supercomputing. *International Journal of Supercomputer Applications*, 10(2) :123–130, 1996.

[DGH+04] M. Daydé, L. Giraud, M. Hernandez, J-.Y. L'Excellent, M. Pantel, and C. Puglisi. An Overview of the GRID-TLSE Project. In *6th International Meeting VECPAR'04 , Valencia, Espagne*, pages 851–856. Universidad Politecnica de Valencia, 2004.

[DHW97] J. Dongarra, S. Hammarling, and D. W. Walker. Key Concepts for Parallel Out-of-Core LU Factorization. *Parallel Computing*, 23 :49–70, 1997.

[DM98] L. Dagum and R. Menon. OpenMP : An Industry Standard API for Shared Memory Programming. volume 5, pages 46–55, 1998.

[Don87] J. J. Dongarra. Performance of Various Computers using Standard Linear Equations Software in a Fortran Environment. In Walter J. Karplus, editor, *Multiprocessors and array processors : proceedings of the Third Conference on Multiprocessors and Array Processors : 14–16 January 1987, San Diego, California*, pages 15–32, San Diego, CA, USA, 1987. Society for Computer Simulation.

[DP] O. Delannoy and S. Petiton. A Peer to Peer Computing Framework : Design and Performance Evaluation of YML. HeteroPar 2004, the Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks. Cork, Ireland, July 2004.

[DPPR] A. Denis, C. Perez, T. Priol, and A. Ribes. Programming the Grid with Distributed Objects. D. Marinescu and C. Lee, editors, Process Coordination and Ubiquitous Computing. CRC Press, 2002.

[DS86] J. Dongarra and D. Sorensen. Linear Algebra on High-Performance Computers. *Applied Mathematics and Computation*, 20(1-2) :57–88, 1986.

[DW95] J. J. Dongarra and D. W. Walker. Software Libraries for Linear Algebra Computations on High Performance Computers. *SIAM Review*, 37(2) :151–180, 1995.

[Fa03] L. Ferreira and al. *Introduction to Grid Computing with Globus*. Redbooks, IBM, 2003.

[FD] G. E. Fagg and J. Dongarra. FT-MPI : Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. Proceeding of the 7th European PVM/MPI Users Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, 2000, pp. 346-353.

[Fed03] G. Fedak. *XtremWeb : une plateforme pour l'étude expérimentale du calcul global pair-à-pair*. Thèse de Doctorat, Université de Paris XI, 2003.

[FGN+] I. Foster, J. Geisler, B. Nickless, W. Smith, and Steven Tuecke. Software Infrastructure for the I-Way High Performance Distributed Computing Experiment. Proceedings of the fifth IEEE Symposium on High Performance Distributed Computing, 1997.

[FKT] I. Foster, C. Kesselman, and S. Tuecke. The Nexus Task-Parallel Runtime System. Proceeding of the first International Workshop on Parallel Processing, 1994.

[Fos01] I. Foster. The Anatomy of the Grid : Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[FTF+02] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G : A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5 :237–246, 2002.

[FTSK] M. Fukuda, Y. Tanaka, N. Suzuki, and S. Kobayashi. A Mobile-Agent-Based PC Grid. Proceeding of the fifth International Workshop on Active Middleware Services, AMS2003. Seattle, WA, June 25, 2003, pages 142–150.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM : Parallel Virtual Machine A Users Guide and Tutorial*

for Networked Parallel Computing. The MIT Press, Cambridge, Massachusetts, 1994.

[GD96] W. Gropp and N. Doss. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6) :789–828, 1996.

[GL04] W. Gropp and E. Lusk. Installation and User's guide to MPICH, a Portable Implementation of MPI Version 1.2.6. Technical report, Mathematic and Computer Science Division. Argonne National Laboratory, University of Chicago, 2004.

[GLT99] W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2 : Advanced Features of the Message Passing Interface.* MIT Press, Cambridge, Massachusetts, 1999.

[GM03] S. Godik and T. Moses. eXtensible Access Control Markup Language (XACML) version 1.0. The OASIS Consortium, 2003.

[GvLPF] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster. Multi-Paradigm Communications in Java for Grid Computing. Communication of the ACM, pages 118–125, October 2001.

[Hen00] D. S. Henty. Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling. SuperComputing 2000. Dallas, USA., 2000.

[HP] B. Hudzia and S. Petiton. Reliable multicast using fault tolerant MPI in the Grid environment. GridNet 2004, San Jose, CA, USA. October 2004.

[KBC+] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore : An Architecture for Global-Scale Persistent Storage. Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.

[KBC+03] S. Krishnamoorthy, G. Baumagartner, D. Cociova, C. Lam, and P. Sadayappan. On Efficient Out-of-Core Matrix Transposition. Technical report, Dept. of Computer and Information Science, The Ohio State University, September 2003.

[KBM02] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software Practice and Experience*, 32(2) :135–164, 2002.

[KBS] J. Ke, M. Burtscher, and E. Speight. Runtime Compression of MPI Messages to Improve the Performance and Scalability of Parallel Applications. High-Performance Computing, Networking and Storage Conference, November 2004.

[KHB+99] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MagPIe : MPI's Collective Communication Operations for Clustered Wide Area Systems. *ACM SIG-PLAN Notices*, 34(8) :131–140, 1999.

[KN94] J. A. Kaplan and L. M Nelson. A Comparison of Queueing, Cluster and Distributed Computing Systems. Technical report, NASA Langley Technical Report Server., 1994.

[KS05] M-T. Kechadi and I. K. Savvas. Dynamic task scheduling for irregular network topologies. *Parallel Computing*, 31(7) :757–776, 2005.

[KTF03] N. Karonis, B. Toonen, and I. Foster. MPICH-G2 : A Grid-Enabled Implementation of the Message Passing Interface. *Parallel and Distributed Computing*, 63 :551–563, 2003.

[LC] S.Y. Lee and C.H. Cho. Load Balancing for Minimizing Execution Time of a Target Job on a Network of Heterogeneous Workstations. In *Lecture Notes in Computer Science*, volume 1911. Proceeding of the 6th Workshop on Job Scheduling Strategies for Parallel Processing.

[LC03] O. Lodygensky and A. Cordier. Auger & XtremWeb : Monte Carlo Computation on a Global Computing Platform. CHEP2003, Computing in High Energy and Nuclear Physics. La Jolla California., 2003.

[Leg03] A. Legrand. *Algorithmique parallèle hétérogène et techniques d'ordonnancement : approches statiques et dynamiques.* Thèse de Doctorat, École Normale Supérieure de Lyon, 2003.

[Ley01] F. Leymann. Web Services Flow Language (WSFL 1.0). IBM Technical Document, 2001.

[LFC+03] O. Lodygensky, G. Fedak, F. Cappello, V. Neri, M. Livny, and D. Thain. XtremWeb & Condor : Sharing Resources Between Internet Connected Condor pools. GP2PC2003, Global and Peer-to-Peer Computing on Large Scale Distributed Systems, Tokyo, May 2003.

[LFL04] M. Langlois, D. Faverio, and M. Lesourd. *XML dans les échanges électroniques : le Framework ebXML.* Hermes, 2004.

[LHO+] S. Langella, S. Hastings, S. Oster, T. Kurc, and U. Catalyurekand J. Saltz. A Distributed Data Management Middleware for Data-Driven Application Systems. The 2004 IEEE International Conference on Cluster Computing, September 2004.

[Li96] Z. Li. *Computational Models and Program for Parallel Out-of-Core Computation.* PhD thesis, Department of Computer Science, Duke University, 1996.

[LT] C. Lee and D. Talia. Grid Programming Models : Current tools, Issues and Directions. In Grid Computing : Making the Global Infrastructure a Reality, F. Berman, G. Fox and T. Hey eds., Wiley, chapt. 21, pp. 555–578, 2003.

[LXC] X. Liu, H. Xia, and A. A. Chien. Network Emulation Tools for Modeling Grid Behavior. The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), May 12-15, 2003 in Tokyo, Japan.

[MB] M. Murshed and R. Buyya. Using the GridSim Toolkit for Enabling Grid Computing Education. International Conference on Communication Networks and Distributed Systems Modeling and Simulation (CNDS 2002), January 27-31, 2002, San Antonio, Texas, USA.

[MDK] T. C. Mowry, A. K. Demke, and O. Krieger. Automatic Compiler-inserted I/O Prefetching for Out-of-Core Applications. Proceedings of the 1996 Symposium on Operating Systems Design and Implementation.

[MTP00] N. Melab, E-G. Talbi, and S. Petiton. A Parallel Adaptive version of the block-based Gauss-Jordan Algorithm. *Journal of supercomputing, Kluwer Academic Publishers*, 17(2) :167–185, September 2000.

[NE01] E. Noulard and N. Emad. A key for Reusable Parallel Linear Algebra Software. *Parallel Computing*, 27(10) :1299–1319, 2001.

[net] Netperf : A Network Performance Benchmark. Technical report, Information Networks Division. Hewlett-Packard Company.

[NHG01] A. Natrajan, M. A. Humphrey, and A. S. Grimshaw. Capacity and Capability Computing Using Legion. *Lecture Notes in Computer Science*, 2073, 2001.

[NKP+00] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems. *International Journal of High Performance Computing Applications, Special Issues on Performance Modelling, Sage Science Press*, 14(3) :228–251, 2000.

[NLRC98] N. Nisan, S. London, O. Regev, and N. Camiel. Globally Distributed Computation over the Internet - The POPCORN Project. In *ICDCS '98 : Proceedings of the The 18th International Conference on Distributed Computing Systems*, 1998.

[NSS99] H. Nakada, M. Sato, and S. Sekiguchi. Design and Implementations of Ninf : towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6) :649–658, 1999.

[PA04] S. Petiton and L. M. Aouad. Large Scale Peer to Peer Performance Evaluations, with Gauss-Jordan Method as an Example. In *Lecture Notes in Computer Science*, volume 3019, pages 938–945. Proceedings of the PPAM 2003, fifth international conference on parallel processing and applied mathematics, September 2004.

[PACa] S. Petiton, L. M. Aouad, and L. Choy. Matrix peer-to-peer computing with very large heterogeneous plateforms. IMACS 2005, 17th World

Congress on Scientific Computation, Applied Mathematics and Simulation.

[PACb] S. Petiton, L. M. Aouad, and L. Choy. Peer to peer large scale linear algebra programming and experimentations. LSSC 2005, fifth International Conference on Large Scale Scientific Computations.

[PB] M. Parashar and J. C. Browne. Conceptual and Implementation Models for the Grid. Proceedings of the IEEE, Special Issue on Grid Computing (2005).

[Pet88] S. Petiton. *Du Dévoloppement de Logiciels Numériques en Environnements Parallèles.* Thèse de Doctorat, Université de Paris 6, 1988.

[PF] S. Pllana and T. Fahringer. Performance Prophet : A Performance Modeling and Prediction Tool for Parallel and Distributed Programs. The 2005 International Conference on Parallel Processing (ICPP-05) ; Performance Evaluation of Networks for Parallel, Cluster and Grid Computing Systems. Oslo, Norway, June 2005.

[PHP+] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. Frank, and C. Chokkareddy. Ocean : The Open Computation Exchange and Arbitration Network, a Market Approach to Metacomputing. Proceedings of the Second International Symposium on Parallel and Distributed Computing, Oct. 2003.

[PKL] P.Wagstrom, S. Krishnan, and G. V. Laszewski. GSFL : A Workflow Framework for Grid Services. Proceeding of the SuperComputing 2002.

[Qui02] M. Quinson. Un outil de prédiction dynamique de performances dans un environnement de metacomputing. *Technique et Science Informatique*, (5), 2002. Numéro spécial RenPar'13.

[Ram00] P. Ramet. *Optimisation de la communication et de la distribution des données pour des solveurs parallèles directs en algèbre linéaire dense et creuse.* Thèse de Doctorat, Université de Picardie Jules Verne, 2000.

[REG+] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond : the OceanStore Prototype. Proceedings of the 2nd USENIX Conference on File and Storage Technologies, FAST'03. March 2003.

[Rei] W. C. Reiley. POOCLAPACK : Parallel Out-of-Core Linear Algebra Package. Technical report, Department of Computer Sciences. The University of Texas at Austin, Nov. 1999.

[RGS+] O. F. Rana, V. S. Getov, E. Sharakan, S. Newhouse, and R. Allan. Developing Grid Services with Jini and JXTA. Global Grid Forum, 2002.

[SAP+96] M. Stonebraker, P. M. Aoki, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa : A Wide-Area Distributed Database System. *VLDB Journal*, 5(1) :48–63, 1996.

[Sar01] L. F. G. Sarmenta. *Volunteer Computing*. PhD thesis, Massachusetts Institute of Technology, 2001.

[SB01] L. Smith and M. Bull. Development of Mixed mode MPI / OpenMP Applications. *Scientific Programming*, 9(2–3) :83–98, 2001.

[SDL+] H. Stockinger, F. Donno, E. Laure, S. Muzaffar, P. Kunszt, G. Andronico, and P. Millar. Grid Data Management in Action : Experience in Running and Supporting Data Management Services in the EU DataGrid Project. Computing in High Energy Physics, CHEP 2003.

[SHTS] M. Sato, M. Hirano, Y. Tanaka, and S. Sekiguchi. OmniRPC : A Grid RPC Facility for Cluster and Global Computing in OpenMP. Workshop on OpenMP Applications and Tools, WOMPAT 2001.

[SK04] I. K. Savvas and M. T. Kechadi. Performance Study of Scheduling Mechanisms for Peer-to-Peer Computing Environments. In *Lecture Notes in Computer Science*, volume 3019, pages 954–962. Proceedings of the PPAM 2003, fifth international conference on parallel processing and applied mathematics., September 2004.

[Ski] D. B. Skillicorn. Motivating Computational Grids. Proceedings of the second IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, 2002.

[SNM+] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. GridRPC : A Remote Procedure Call API for Grid Computing. GRID 2002, the Third International Workshop, Baltimore, MD, USA, November 18, 2002, LNCS 2536.

[SOHL+96] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI : The Complete Reference*. The MIT Press, Cambridge, Massachusetts, 1996.

[SPBT] A. Sulistio, G. Poduvaly, R. Buyya, and C.-K. Tham. Constructing A Grid Simulation with Differentiated Network Service Using GridSim. Proceeding of the 6th International Conference on Internet Computing (ICOMP'05), June 27-30, 2005, Las Vegas, USA.

[Sut02] F. Suter. *Parallélisme mixte et prédiction de performances sur réseaux hétérogènes de machines parallèles.* Thèse de Doctorat, École Normale Supérieure de Lyon, 2002.

[SYAD] K. Seymour, A. YarKhan, S. Agrawal, and J. Dongarra. NetSolve : Grid Enabling Scientific Computing Environments. Grid Computing and New Frontiers of High Performance Processing, Grandinetti, L. eds. Elsevier, 2005.

[TAA+] B. Traversat, A. Arora, M. Abdelziz, M. Duigou, and C. Haywood. Project JXTA 2.0 Super-Peer Virtual Network. Sun Microsystems Inc., 2003.

[Tha01] S. Thatte. XLANG : Web services for business process design. Microsoft Technical Document, 2001.

[TKV+] A. R. Tripathi, N. M. Karnik, M. K. Vora, T. Ahmed, and R. D. Singh. Mobile Agent Programming in Ajanta. Proceedings of the 19th International Conference on Distributed Computing Systems, Austin, Texas, pp. 314–322, 1999.

[TMN+] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 97-104, 1999.

[Tol99] S. Toledo. A Survey of Out-of-Core Algorithms in Numerical Linear Algebra. In *American Mathematical Society*, pages 161–179. In James M. Abello and Jeffrey Scott Vitter, editors, External Memory Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1999.

[TTL02] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing : Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., 2002.

[TTL04] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice : The Condor Experience. *Concurrency and Computation : Practice and Experience*, 2004.

[use] *Network Weather Service User's Guide.*

[VGLP] S. Verma, J. Gawor, G. V. Laszewski, and M. Parashar. A CORBA Commodity Grid Kit. In Second International Workshop on Grid Computing - GRID 2001. Lecture Notes in Computer Science, C. A. Lee, Ed., vol. 2241. Denver : Springer, Nov. 2001, pp. 212.

[VNRS] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov. Framework for Peer to Peer Distributed Computing in a Heterogeneous, Decentralized Environment. GRID 2002, the Third International Workshop on Grid Computing. Baltimore, MD, USA.

[WHH+92] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn : A Distributed Computational Economy. *Software Engineering*, 18(2) :103–117, 1992.

[Wol] Rich Wolski. Experiences with Predicting Resource Performance Online in Computational Grid Settings. ACM SIGMETRICS Performance Evaluation Review, Volume 30, Number 4, pp 41–49, March, 2003.

[WPBB01] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. G-commerce : Market Formulations Controlling Resource Allocation on the Computational

Grid. In *International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, April 2001.

[WSH99] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service : A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5–6) :757–768, 1999.

[WSO01] N. Wang, D. C. Schmidt, and C. ORyan. An Overview of the CORBA Component Model. Component-Based Software Engineering : Putting the Pieces Together, G. T. Heineman and W. T. Councill, Eds., 2001.

[XK] X. Xu and M. Kezunovic. Mobile Agent Software Applied in Maintenance Scheduling. North American Power Symposium - NAPS, College Station, Texas, October 2001.

[YTS] N. Yamamoto, O. Tatebe, and S. Sekiguchi. Parallel and Distributed Astronomical Data Analysis on Grid Datafarm. GRID 2004, fifth IEEE/ACM international workshop on Grid Computing. SuperComputing 2004, Pittsburgh.

[Zhu04] Y. Zhu. A Survey on Grid Scheduling Systems. Technical report, Department of Computer Science, Hong Kong University of Science and Technology, 2004.

[ZLC95] M. J. Zaki, W. Li, and M. Cierniak. Performance Impact of Processor and Memory Heterogeneity in a Network of Machines. Technical report, University of Rochester, 1995.

[ZS] X. Zhang and J. Schopf. Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. Proceedings of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Workshop (IPCCC), April 2004.

Table des figures

Liste des tableaux

Résumé

Les grilles de calcul offrent une alternative intéressante pour les applications de calcul matriciel, grandes consommatrices de ressources de calcul et de mémoire. Néanmoins, les modèles de calcul de type 'task farming' inhérents à la plupart de ces systèmes rendent difficile une programmation efficace car la granularité de ces applications les rend inadaptées. Dans cette thèse, nous proposons un modèle de programmation sur grilles de calcul non-dédiées basé sur la gestion de la localité des données ; du réseau de communication jusqu'aux mémoires locales des noeuds de calcul, pour des applications matricielles basées sur des distributions par blocs. La technique de programmation *out-of-core* est introduite comme réponse aux problèmes de restriction mémoire sur les noeuds de calcul. Cette technique tend à minimiser l'impact des entrées/sorties nécessaires au calcul lorsque la taille des tâches excède la taille mémoire allouable. D'un autre côté, des techniques de placement persistant proposant l'anticipation des migrations et le clouage des données permettent d'optimiser les besoins en communications et d'atteindre un bon niveau de performances par rapport aux implémentations classiques. Les évaluations de performances ont été réalisées sur deux noyaux de l'algèbre matriciel sur de larges plateformes de calcul déployées sur trois sites géographiquement distribués, en France et au Japon, et sur la plateforme expérimentale Grid'5000.

Abstract

Grid computing provides an interesting alternative for matrix algebra applications which need very large resources of calculation and memory. However, the inherent task farming programming model in most of these systems makes effective programming more difficult because of the granularity of these applications, which makes them unsuited for the Grid. This thesis presents a well-adapted programming model for distributed and parallel computation for a block-based matrix algebra applications on non-dedicated Grid platforms based on data locality ; for the communications network to the local memories on the computational nodes. The out-of-core programming is introduced as a response to restrictions and constraints of memory on the computational nodes. This technique aims to minimize the impact of I/O needs when the size of tasks exceeds the size of available memory. On the other hand, persistent data placement techniques including pre-deployment, migration anticipation and data referencing allow to optimize the communications needs and to reach a good level of performance compared to the classical implementations. The performance evaluation were made and discussed for two basic linear algebra operations on large platforms deployed on three geographically distributed sites, in France and in Japan, and on the Grid'5000 experimental platform.