

# Mémoire

Préparé au

**Laboratoire d'Analyse et d'Architecture des Systèmes du  
CNRS**

En vue de l'obtention de

L'HABILITATION À DIRIGER DES RECHERCHES DE L'INSTITUT NATIONAL  
POLYTECHNIQUE DE TOULOUSE  
Spécialité INFORMATIQUE

par

Thierry MONTEIL

Titre du mémoire :

*Du cluster à la grille sous l'angle de la performance*

Soutenue le 7 décembre 2010 devant le jury

M. :	Michel	DAYDÉ	Président
MM. :	Frédéric	DESPREZ	Rapporteurs
	Emmanuel	JEANNOT	
	Pierre	SENS	
MM. :	Jean Marie	GARCIA	Examineurs
	Christophe	CHASSOT	

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Evolution des plates-formes . . . . .	1
1.1.1	Les grappes . . . . .	1
1.1.2	Les grilles . . . . .	1
1.1.3	Les nuages . . . . .	4
1.2	Modèle en couche . . . . .	4
1.3	Plan du mémoire . . . . .	4
<b>2</b>	<b>Couche fabrique</b>	<b>7</b>
2.1	Introduction . . . . .	7
2.2	Contribution sur la simulation fine de l'utilisation de ressources par des appli- cations parallèles . . . . .	8
2.2.1	Contexte . . . . .	8
2.2.2	La problématique traitée . . . . .	9
2.2.3	Le modèle proposé . . . . .	9
2.2.4	Quelques résultats . . . . .	12
2.3	Contribution sur la prédiction de charge analytique . . . . .	13
2.3.1	Contexte . . . . .	13
2.3.2	La problématique traitée . . . . .	13
2.3.3	Le modèle proposé . . . . .	13
2.3.4	Quelques résultats . . . . .	15
2.4	Contribution sur la localisation du stockage de données dans les bibliothèques de passage de messages . . . . .	17
2.4.1	Contexte . . . . .	17
2.4.2	La problématique traitée . . . . .	17
2.4.3	Résolution sur un cas d'école . . . . .	18
2.4.4	Quelques résultats . . . . .	20
2.5	Contribution sur la reconfiguration du réseau pour minimiser la consommation énergétique sous des contraintes de QoS . . . . .	21
2.5.1	Contexte . . . . .	21
2.5.2	La problématique traitée . . . . .	22
2.5.3	Le modèle proposé . . . . .	22
2.5.4	Quelques résultats . . . . .	27
2.6	Bilan . . . . .	28

<b>3</b>	<b>La couche service</b>	<b>29</b>
3.1	Introduction . . . . .	29
3.2	Contributions sur l'ordonnancement . . . . .	29
3.2.1	Les algorithmes . . . . .	29
3.2.2	Ordonnancement et qualité de service . . . . .	30
3.2.3	Ordonnancement et réseau . . . . .	35
3.2.4	Ordonnancement et modèle économique . . . . .	38
3.3	Contribution sur l'observation . . . . .	41
3.3.1	Contexte . . . . .	41
3.3.2	La problématique traitée . . . . .	41
3.3.3	Le système proposé . . . . .	42
3.4	Bilan . . . . .	45
<b>4</b>	<b>La couche intergiciel</b>	<b>47</b>
4.1	Introduction . . . . .	47
4.2	Contribution sur les environnements de programmation parallèle . . . . .	47
4.2.1	Introduction . . . . .	47
4.2.2	Présentation des différents modules de LANDA . . . . .	47
4.2.3	Les fichiers . . . . .	48
4.2.4	Le noyau . . . . .	48
4.2.5	Les interfaces graphiques . . . . .	48
4.2.6	Utilisation de la bibliothèque . . . . .	50
4.2.7	Ma contribution . . . . .	50
4.3	Contributions sur les environnements pour les fournisseurs de service . . . . .	50
4.3.1	Introduction . . . . .	50
4.3.2	La notion de contrat . . . . .	50
4.3.3	La sécurité . . . . .	51
4.3.4	Interactions entre un client et le gestionnaire de services . . . . .	52
4.3.5	Ma contribution . . . . .	54
4.4	Contribution sur les environnements autonomiques . . . . .	54
4.4.1	Contexte . . . . .	54
4.4.2	Le logiciel TUNe . . . . .	54
4.4.3	Les politiques de gestion (PD) . . . . .	57
4.4.4	Quelques résultats . . . . .	58
4.5	Bilan . . . . .	59
<b>5</b>	<b>La couche application</b>	<b>61</b>
5.1	Introduction . . . . .	61
5.2	Code de simulation électromagnétique sur grille . . . . .	61
5.2.1	Contexte . . . . .	61
5.2.2	Déploiement et gestion des erreurs avec TUNe . . . . .	62
5.2.3	Pré-étude sur le passage sur grille à grande échelle . . . . .	64
5.3	Analyse de code . . . . .	67
5.3.1	Contexte . . . . .	67
5.3.2	La problématique traitée . . . . .	68
5.3.3	Le modèle proposé . . . . .	68
5.4	Bilan . . . . .	73

<b>6 Synthèse et perspectives</b>	<b>75</b>
6.1 Résumé des contributions . . . . .	75
6.2 Vision et Perspectives de recherche . . . . .	77
6.2.1 Evolution des ressources matérielles . . . . .	77
6.2.2 Evolutions des pratiques et des mentalités . . . . .	78
6.2.3 Evolution de la complexité . . . . .	79
6.2.4 Quelques projets à venir . . . . .	79
<b>Bibliographie</b>	<b>82</b>



# Table des figures

2.1	Application maître/esclaves . . . . .	10
2.2	Modèle serveur . . . . .	11
2.3	Application moyen grain, esclaves lents . . . . .	12
2.4	Application moyen grain, maître lent . . . . .	13
2.5	Modèle “round-robin” multi-serveurs . . . . .	14
2.6	Exemple de la chaîne de Markov lorsque $d < C$ . . . . .	14
2.7	Moyenne du nombre de processus dans le système . . . . .	16
2.8	Approximation du temps d’exécution d’un processus . . . . .	16
2.9	choix du routage en fonction de la charge . . . . .	20
2.10	Puissance consommée en fonction du nombre d’applications et de $\alpha$ . . . . .	27
3.1	Modèle de files d’attente . . . . .	31
3.2	L’algorithme de placement proposé . . . . .	33
3.3	Exemple de création du graphe de l’application avec précedence à partir d’un graphe de tâches . . . . .	36
3.4	Exemple simple de demande de placement au niveau grille . . . . .	37
3.5	Simulation des placements des applications avec prise en compte d’aspect économique . . . . .	40
3.6	Schéma d’utilisation d’Aroma . . . . .	42
3.7	Serveur générique . . . . .	43
4.1	Fenêtre principale de LANDA . . . . .	49
4.2	Interface graphique cliente . . . . .	53
4.3	HD pour la description matérielle pour Grid’5000 . . . . .	55
4.4	SD pour une architecture DIET . . . . .	56
4.5	Méta-modèle des diagrammes PD . . . . .	57
4.6	Exemple d’expression des opérateurs . . . . .	57
4.7	PD pour l’optimisation de l’architecture de DIET . . . . .	58
4.8	Capacité des PD à s’autogénérer . . . . .	58
4.9	Gestion de la capacité de traitement dans DIET . . . . .	59
5.1	Description en TUNE de Yatpac . . . . .	63
5.2	Description de la réparation d’une simulation . . . . .	63
5.3	Quelques résultats de TUNE et Yatpac . . . . .	64
5.4	Estimation du speed-up dans un découpage en tranches . . . . .	66
5.5	Estimation du speed-up dans un découpage en pavés . . . . .	67

5.6	Comparaison entre le temps d'exécution estimé et le temps d'exécution réel du programme . . . . .	69
5.7	Comparaison entre le nombre d'exécutions des blocs de base réel et celui estimé par le modèle de prédiction . . . . .	71
5.8	Comparaison des erreurs obtenues sans et avec annotations . . . . .	73
6.1	Synthèse des travaux . . . . .	77
6.2	Exemple d'architecture d'un méso-centre de calcul . . . . .	78

# Chapitre 1

## Introduction

Le développement très dynamique de l'Internet et plus généralement des réseaux de communication a permis la naissance de nouvelles formes d'utilisation de l'informatique durant ces dix dernières années. Tout d'abord au niveau matériel, ceci a ouvert la porte à la création d'architecture distribuée géographiquement [54] ensuite au niveau logiciel ceci a offert l'opportunité de concevoir de nouveaux modes d'utilisation.

Ce mémoire abordera différentes contributions dans les couches logiciels nécessaires à l'utilisation d'une plate-forme distribuée pour mettre en place de nouveaux usages.

### 1.1 Evolution des plates-formes

#### 1.1.1 Les grappes

Une grappe de machine (Cluster en Anglais) désigne un ensemble de d'ordinateurs, appelés nœuds, tous inter-connectés, dans le but de partager des ressources informatiques. Une grappe peut être constituée d'ordinateurs de bureaux, de "racks" de machines constituées de composants standards ou de "lames" également constituées de composants standards afin d'optimiser l'espace physique. Une grappe est généralement composée de machines homogènes en termes d'architecture et de système d'exploitation. Elle ne regroupe que des machines appartenant au même domaine d'administration réseau et les nœuds communiquent entre eux en utilisant un réseau de communication rapide. Les différents nœuds d'une grappe possèdent souvent une configuration logicielle semblable. Une pratique courante, utilisée par la plupart des logiciels d'administration de grappes (Rocks Cluster Distribution[79], IBM CSM[56], SUN Cluster[88]) est d'installer les logiciels sur un nœud maître, puis de déployer une image du système sur chaque nœud de la grappe.

#### 1.1.2 Les grilles

Le terme grille désigne un ensemble beaucoup plus important de machines, hétérogènes, réparties sur différents domaines d'administration réseau. Les différentes entités composant une grille peuvent être réparties sur l'ensemble de la planète et communiquent entre elles en utilisant une grande diversité de réseaux allant de l'Internet à des réseaux privés très haut débit.

Le concept de grille informatique puise son inspiration dans le développement des grilles d'électricité (*power grids*) au début du XXIème siècle [63]. A cette époque, la révolution ne

résidait pas en l'électricité elle-même, mais plutôt en la constitution d'un réseau électrique fournissant aux individus un accès fiable et peu onéreux à l'électricité, au travers d'une interface standard : la prise de courant [54]. Les composants formant le réseau électrique sont hétérogènes, et la complexité induite est totalement masquée à l'utilisateur final. Ainsi, une grille informatique possède les mêmes propriétés d'hétérogénéité des ressources que le réseau électrique, le défi scientifique est d'offrir à l'utilisateur de la grille la même transparence d'utilisation qu'offre la prise électrique à l'utilisateur du réseau électrique.

Une grille informatique est une infrastructure matérielle et logicielle qui fournit un accès consistant et peu onéreux à des ressources informatiques [54]. Le but est ainsi de fédérer des ressources provenant de diverses organisations désirant collaborer en vue de faire bénéficier aux utilisateurs d'une capacité de calcul et de stockage qu'une seule machine ne peut fournir. Cependant, tout système informatique distribué ne peut posséder l'appellation de grille [53]. En effet, une grille est un système qui coordonne des ressources non soumises à un contrôle centralisé, qui utilise des protocoles et interfaces standards dans le but de délivrer une certaine qualité de service (en termes de temps de réponse ou bien de fiabilité par exemple).

Plusieurs types de grilles peuvent être discernées selon l'utilisation recherchée :

- *Grille d'information* : la ressource partagée est la connaissance. L'Internet en est le meilleur exemple : un grand nombre de machines hétérogènes réparties sur toute la surface du globe autorisant un accès transparent à l'information.
- *Grille de stockage* : l'objectif de ces grilles est de mettre à disposition un grand nombre de ressources de stockage d'information afin de réaliser l'équivalent d'un "super disque dur" de plusieurs PetaBytes. Le projet DataGrid ou les réseaux Kaaza ou Gnutella sont un bon exemple de grille de stockage.
- *Grille de calcul* : l'objectif de ces grilles est clairement d'agréger la puissance de traitement de chaque nœud de la grille afin d'offrir une puissance de calcul "illimitée". Mes travaux se situent plutôt pour ce type de grille.

Le calcul sur grille possède plusieurs objectifs [90] :

- *Exploiter les ressources sous-utilisées* : Dans la plupart des organisations, il existe une quantité très importante de ressources sous-utilisées. Des études montrent que le taux d'utilisation d'un ordinateur de bureau n'atteint pas les 5 % de moyenne. Ainsi, il est intéressant de pouvoir utiliser ces ressources libres pour exécuter une application lorsque les machines qui lui sont normalement dédiées ne peuvent le faire dans de bonnes conditions, en cas de pics d'utilisation par exemple.
- *Fournir une importante capacité de calcul parallèle* : Le principal intérêt d'une grille est de permettre l'exécution de plusieurs tâches en parallèle. Ainsi, une application pouvant être découpée en plusieurs tâches, pourra être exécutée sur plusieurs machines de la grille, réduisant ainsi le temps de réponse du calcul à effectuer.
- *Accéder à des ressources additionnelles* : Outre les processeurs et les capacités de stockage, l'utilisation d'une grille peut s'avérer utile pour accéder à d'autres types de ressources, tels que des équipements spéciaux, des logiciels, et bien d'autres services. Certaines machines peuvent, par exemple, héberger des logiciels ayant des coûts de licence très élevés.
- *Mieux répartir l'utilisation des ressources* : Puisqu'une grille de calcul permet d'exécuter des applications sur des machines inactives, il est possible de répartir des pics

d'utilisation inattendus de certaines machines vers d'autres qui sont moins sollicitées.

- *Gérer des applications avec deadline proche* : Si une application doit être exécutée avec une contrainte de date butoir très proche, l'utilisation d'une grille peut s'avérer utile. En effet, si l'application peut être découpée en un nombre suffisant de tâches, et si une quantité adéquate de ressources peut lui être dédiée, l'application bénéficiera d'une capacité de calcul suffisante pour être exécutée tout en respectant une *deadline* proche.
- *Assurer une tolérance aux fautes pour un coût moindre* : Dans les systèmes conventionnels, la tolérance aux fautes est réalisée grâce à la redondance du matériel sensible. Cette solution possède l'inconvénient d'avoir un coût assez élevé. Les grilles de calcul, de par leur nature, offrent une solution alternative pour effectuer de la tolérance aux fautes. En effet, si une défaillance apparaît sur un site de la grille, les autres parties de la grille ne seront pas forcément affectées. Ainsi, des données peuvent être dupliquées sur plusieurs machines de la grille pour prévenir leur perte en cas de défaillance. De plus, pour des applications temps réel critiques, il peut s'avérer utile d'en exécuter plusieurs instances simultanément sur différentes machines, voire même de vérifier les résultats qu'elles fournissent pour plus de sûreté.

Les spécificités des grilles de calcul rendent leur gestion délicate [64] de par :

- *L'hétérogénéité* : la première caractéristique des ressources constituant une grille est sans nul doute leur hétérogénéité. Qu'elles soient matérielles ou bien logicielles, les ressources sont souvent très différentes les unes des autres. Cette hétérogénéité impose des contraintes de portage de code, d'utilisation de langages multi-plateformes et d'utilisation de protocoles de communication standardisés. L'utilisateur doit de plus pouvoir utiliser la grille de manière transparente et homogène quelle que soit l'architecture de sa propre machine et l'architecture de la machine à laquelle il se connecte.
- *La multiplicité des domaines d'administration* : les ressources sont géographiquement distribuées et appartiennent à différentes organisations indépendantes. Elles appartiennent donc à plusieurs domaines d'administration distincts, ayant chacun sa propre politique de gestion et de sécurité en terme d'accès au réseau, d'accès aux données, d'authentification ou encore de confidentialité. Ainsi, les personnes chargées d'administrer la grille n'auront pas forcément de privilèges particuliers sur les machines des différents domaines d'administration. Il est donc indispensable de mettre au point des méthodes d'administration particulières ne nécessitant aucun privilège sur les machines cibles. Il est également indispensable que chaque administrateur local conserve ses propres privilèges sur les machines qui lui appartiennent.
- *L'aspect dynamique* : du fait du grand nombre de ressources considérées, la défaillance d'une ressource (machine ou réseau) est un événement courant qui ne doit pas mettre en péril le fonctionnement de la grille. Le gestionnaire de ressources tout comme les applications doivent tenir compte de cet aspect et être capables de réagir rapidement à la perte ou à l'ajout d'un nœud. De la même manière, l'ajout de fonctionnalités logicielles doit pouvoir se faire, dans la mesure du possible, sans qu'il soit nécessaire d'arrêter ou de réinstaller l'ensemble des nœuds de la grille.
- *La gestion des ressources* : la capacité de mise à l'échelle d'une grille est également une caractéristique à prendre en compte. En effet, une grille pourra être constituée d'une dizaine de ressources, tout comme de plusieurs milliers de ressources. Ce problème de dimensionnement pose de nouvelles contraintes sur les applications et les algorithmes de

gestion des ressources. L'observation des ressources notamment, ne doit pas induire une sur-consommation de ressources trop importante et ce quelque soit le nombre de nœuds de la grille.

### 1.1.3 Les nuages

Le concept de nuage ou "cloud" [73] est une évolution de la notion de grille. Il se focalise plus sur les fournisseurs des ressources qui mettront à disposition la puissance de calcul et de stockage avec leur propre technologie. La définition d'utilisateurs et de fournisseurs permet de créer une relation claire entre ces deux types d'entités. On pourra alors associer des garanties de puissance fournie ou de quantité de stockage par exemple. Trois modèles peuvent être utilisés sur un nuage :

- *IaaS : Infrastructure as a service*, le fournisseur met à disposition uniquement des ressources matérielles (machine, réseaux, disque), l'utilisateur gère le système, les intergiciels et les applications.
- *PaaS : Platform as a service*, l'utilisateur ne gère et ne contrôle ici que ses applications métier, le reste est délégué au fournisseur de service
- *SaaS : Software as a service*, c'est l'ultime niveau de transfert vers le fournisseur, l'utilisateur ne gère plus que ses données métier, il utilise les applications fournies par le fournisseur. Il s'agit d'utiliser le logiciel à la demande.

## 1.2 Modèle en couche

Les grilles se prêtent assez bien à une modélisation en couches de leur architecture [63]. Selon la grille et leur philosophie (grille de recherche ou grille de production par exemple), plusieurs représentations sont possibles [24, 55, 94, 1]

Dans la suite de ce mémoire, le modèle suivant [57] servira de fil conducteur en partant de la couche basse :

- *La couche fabrique* contient des protocoles et des interfaces qui permettent d'accéder à des ressources physique ou logique de la grille. Cette couche contient entre autres les systèmes d'exploitation, les bibliothèques systèmes, les protocoles réseaux, etc.
- *La couche service* définit des protocoles, des services de base pour faire des actions sur la grille. Dans cette couche, on retrouve : le système pour la sécurité (comme GSI : Grid Security Infrastructure), la gestion des ressources (comme GRAM : Grid Resource Allocation Management ou GRIP : Grid Resource Information Protocol), etc.
- *La couche intergiciel* qui sera le point d'accès pour les utilisateurs. Elle contient les compilateurs parallèles, les environnements de programmation sur grille, les outils de haut niveau pour le déploiement d'application, les environnements d'accès distant, etc.
- *La couche Application* contient les logiciels développés par l'utilisateur dans différents domaines : WEB, simulation, calcul, etc.

## 1.3 Plan du mémoire

L'ensemble des travaux présentés concerne les aspects performance au sens large en terme : d'exécution d'application parallèle, de programmation parallèle, de consommation énergétique

et de coût financier. Le mémoire sera décomposé en six chapitres dont quatre dédiés aux contributions apportées dans le domaine des clusters et des grilles :

- Le chapitre 1 introduit le domaine ciblé en posant quelques problèmes ainsi que la structuration du mémoire à travers les différents travaux sur les couches de l’architecture d’une grille.
- Le chapitre 2 s’intéresse à la couche la plus basse : la fabrique. Quatre contributions liées à cette couche sont décrites : la modélisation et la simulation fines des ressources d’une grappe (processeur, réseau, mémoire, système, etc) de manière discrète puis dans le cas du processeur de manière stochastique en vue de leurs utilisations dans des applications distribuées, le problème de la localisation des messages et de leur utilisation des ressources dans le cadre des bibliothèques de passage de messages et enfin la configuration du coeur de réseau pour garantir une qualité de service et une faible consommation énergétique.
- Le chapitre 3 traite de la couche service en s’intéressant à deux problématiques le service d’observation de l’état de la plate-forme et les problèmes d’ordonnancement. Pour ce dernier, trois aspects sont couverts : la qualité de service, la prise en compte du réseau et l’insertion de modèle économique.
- Le chapitre 4 couvre la couche intergiciel avec trois outils : un environnement de programmation parallèle (LANDA), un environnement de mise en place du modèle ASP (Application Service Provider) avec le logiciel AROMA et enfin une contribution à l’environnement de déploiement et de gestion autonome TUNe.
- Le chapitre 5 aborde la couche application avec deux travaux un sur un code de simulation électromagnétique et sa gridification et un autre sur l’analyse de code applicatif.
- le chapitre 6 conclut ce mémoire et donne des perspectives sur les travaux futurs.



## Chapitre 2

# Couche fabrique

### 2.1 Introduction

La couche fabrique joue un rôle primordial dans la gestion de la performance. En effet c'est elle qui fournit les ressources de base qui serviront à exécuter les applications. Il y a plusieurs difficultés :

- la grande diversité de ressources à prendre en compte : matériel, logiciel
- le fonctionnement totalement différent des nombreux types de ressources, par exemple un processeur n'a rien à voir en terme de fonctionnement avec un système de fichiers
- le besoin de tenir compte de la dépendance créée entre les ressources lors de leur utilisation dans le contexte des grilles.

Sous l'angle de la performance, comprendre l'utilisation des ressources fait par la couche fabrique permet d'analyser le déroulement des programmes et éventuellement de prédire ce qu'il va se passer lors de l'exécution d'un programme. Ceci peut alors alimenter un ordonnanceur de la grille ou bien encore permettre de faire de la planification de ressources afin d'anticiper l'extension ou la re-configuration d'une plate-forme (grille, data-center).

La première contribution décrite dans ce chapitre concerne la capacité à simuler finement l'utilisation des ressources par une application distribuée.

La deuxième contribution envisage le même type de problème mais en le traitant de manière stochastique et analytique, l'objectif est alors de prédire analytiquement le temps d'exécution d'un processus sur une machine multi-processeurs où la charge non liée à l'application est représentée de manière probabiliste.

La troisième contribution se focalise sur les problèmes d'échange de messages entre des applications communicantes et plus particulièrement sur la corrélation qu'il existe entre l'utilisation des ressources et le chemin que suit un message d'un émetteur vers un récepteur et de l'impact que cela a sur la performance de l'application.

La dernière contribution fait le lien entre les besoins en qualité de service des applications en terme de réseau et la configuration nécessaire des routeurs constituant le coeur de réseau sous des contraintes d'économie d'énergie.

## 2.2 Contribution sur la simulation fine de l'utilisation de ressources par des applications parallèles

### 2.2.1 Contexte

Une question récurrente en calcul parallèle ou en mise à disposition de service est : ai je assez de ressources pour faire les calculs que je souhaite ou pour déployer les services que je veux utiliser ? Cette question n'est pas triviale car d'une part ce sont des applications distribuées sur plusieurs ressources et d'autre part il peut y avoir un nombre conséquent d'applications se déroulant simultanément. Un phénomène de mise en concurrence apparaît au niveau de la machine (processeur, mémoire, disque, etc), du réseau ou bien encore du système d'exploitation. Pour tenter de répondre à cette question, il faut connaître le déroulement des applications, le fonctionnement du matériel et des différentes couches logicielles qui vont intervenir. Il s'ajoute à tout cela éventuellement le comportement humain s'il s'agit d'applications interactives. Différentes techniques sont possibles :

- **les benchmarks** : Les techniques de benchmarks reposent sur des analyses métrologiques d'applications réelles ou d'applications spécifiquement créées pour l'évaluation de performances [84]. Elles consistent à exécuter une application de référence sur une machine réelle et à mesurer les performances du couple application/machine. Ces techniques s'avèrent le plus souvent très précises puisqu'elles reposent sur une exécution réelle de l'application, mais sont également délicates à mettre en œuvre. Elles nécessitent de disposer à la fois des machines support d'exécution, et d'instrumenter l'application afin d'obtenir des indices de performance pertinents. Le délai d'obtention des résultats peut également représenter un frein à l'utilisation de ces techniques. Ce délai correspond en effet, au temps réel d'exécution de l'application, qui est le plus souvent non négligeable. De plus, il est toujours délicat de certifier que l'instrumentation du système étudié ne modifie pas le comportement temporel de ce système, falsifiant ainsi les résultats obtenus.
- **la simulation** : Simuler un système signifie construire un modèle du comportement de ce système, et étudier le comportement de ce modèle lorsqu'il est soumis à une certaine charge de travail [85][74][75][45]. La simulation possède l'avantage de pouvoir s'appliquer à tous types de problèmes. Cependant certaines précautions doivent être prises afin de garantir l'efficacité de cette technique :
  - Il est nécessaire de définir les parties du système à simuler et avec quel niveau de détails. Il est en effet inutile de perdre du temps à détailler le comportement de certaines parties du système si ces dernières n'ont qu'une faible influence sur le résultat qui nous intéresse. Un trop grand niveau de détails augmente également le risque d'erreurs.
  - Un grand nombre d'informations peut être fourni par une simulation. Il est nécessaire de cibler les informations que l'on souhaite obtenir afin de les extraire, par des méthodes statistiques par exemple, de la masse d'information fournie par la simulation.
  - Le modèle doit pouvoir être implémenté le plus efficacement possible afin de garantir un temps de simulation le plus court possible.
- **les méthodes analytiques** : elles possèdent l'avantage d'être rapides et d'offrir une vision simplifiée du comportement du système. Cependant l'ensemble des problèmes modélisables de manière totalement analytique est de taille réduite. De ce fait, représenter le comportement détaillé d'un système devient rapidement impossible. Il existe différentes

méthodes :

- modélisations stochastiques [61][31][59], ces études se limitent à des systèmes de taille réduite (une seule machine) ou à l'étude de phénomènes très précis (modélisation fine du comportement d'un disque dur, par exemple).
- méthodes "historiques" développées plus récemment [3][49][7] dans le but d'aider à l'ordonnancement de tâches dans les grilles de calcul. Ces méthodes "historiques" reposent sur l'analyse détaillée de mesures de performance issues de traces réelles d'exécutions d'applications sur différentes machines et avec différents paramètres d'entrée. L'objectif est de discerner les éléments de l'application, ou de la machine, ayant un impact significatif sur les performances du système, afin de déterminer un ensemble d'équations permettant de prédire efficacement la durée d'exécution d'une application sur une machine donnée.
- Méthode Layered Queuing Networks [80] est une extension des réseaux de files d'attente dont le but est de permettre la modélisation du phénomène de possession simultanée de ressources. Cette méthode se base sur une représentation des applications et des machines sous forme de tâches organisées en couches [47]. Chaque tâche représente, soit un traitement particulier d'une application, soit une ressource physique (processeur, disque ...) et est modélisée par une file d'attente. Une tâche ne peut requérir le service que d'une autre tâche de niveau inférieur. Ce service peut être une requête synchrone, asynchrone ou à plusieurs phases (durant une première phase, la requête est synchrone, puis durant la seconde la tâche appelante est libérée alors que la tâche appelée doit réaliser un traitement supplémentaire).

## 2.2.2 La problématique traitée

Notre objectif est de simuler le fonctionnement de grappes de machines utilisées en mode ASP (Application Service Provider). Le système modélisé est composé de plusieurs serveurs reliés entre eux par des réseaux d'inter-connexion. Sur chacun de ces serveurs, plusieurs instances d'applications, ayant des caractéristiques propres peuvent s'exécuter simultanément. Un nœud particulier de la grappe joue le rôle de frontal du système ASP. Il répartit les applications sur les différents serveurs de la grappe, en utilisant une politique d'ordonnancement donnée, afin d'optimiser la qualité de service globale du système ASP.

Le but est alors de pouvoir dimensionner les grappes de machines et de pouvoir visualiser le comportement du système face à un changement d'utilisation ou de prédire l'impact d'une modification d'un des composants des grappes de machines sur les performances globales du système.

Différents types d'applications sont concernés par ce type d'environnements : des applications de calcul (séquentielles ou parallèles) et des serveurs applicatifs donnant accès à des contenus multimédias interactifs (serveurs webs de pages statiques et dynamiques, et serveurs d'applications multi-tiers). A titre d'exemple dans ce mémoire, seul les résultats sur la modélisation d'applications de calcul sera explicitée.

## 2.2.3 Le modèle proposé

La modélisation de l'ensemble des acteurs intervenant va être décrit dans les paragraphes suivants.

### 2.2.3.1 Les applications

Les cas des applications représentables sous forme de graphe et des applications maître/esclaves ont été traités. Plus précisément pour le deuxième cas, le travail effectué par chacun des acteurs est le suivant :

pour le maître:

- tant qu'il y a des valeurs à calculer
  - envoi de points à calculer aux esclaves en attente de travail
  - récupération des valeurs calculées par les esclaves
  - calcul de résultats intermédiaires

pour un esclave :

- boucle
  - attente de travail
  - calcul
  - envoi de résultat

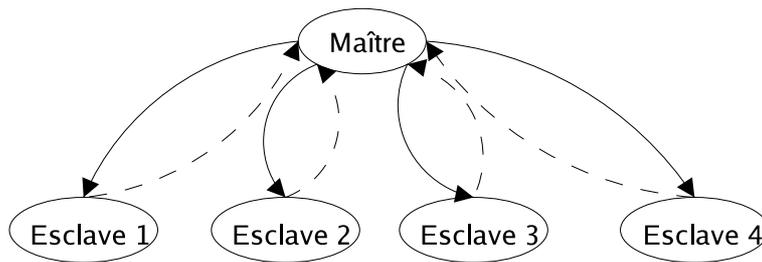


FIGURE 2.1 – Application maître/esclaves

Chacune des tâches (maître et esclaves) réalise un traitement similaire à celui d'une application séquentielle, cependant, toutes les tâches esclaves ont les mêmes caractéristiques. En résumé, une application maître/esclave est caractérisée par :

- le nombre de tâches (maître + esclaves),
- le nombre d'itérations de calcul à réaliser,
- les distributions associées à la tâche maître,
- les distributions associées aux tâches esclaves.

### 2.2.3.2 L'intergiciel

Les messages échangés par les applications parallèles utilisent des bibliothèques de passage de messages telles que MPI [42] ou PVM [46]. Ces bibliothèques définissent plusieurs modes de communication tels que les communications bloquantes, synchrones ou bufferisées. Ces communications peuvent utiliser plusieurs moyens de communication (différents protocoles de transport, mémoire partagée ...). Dans le cadre de notre étude, nous avons simulé les communications bloquantes de la norme MPI telles qu'elles sont implémentées dans LAM/MPI [15][86]. Chaque bibliothèque de passage de messages ayant ces particularités, il est important

de bien modéliser cela. Dans LAM/MPI, le système de tampon est important à prendre en compte.

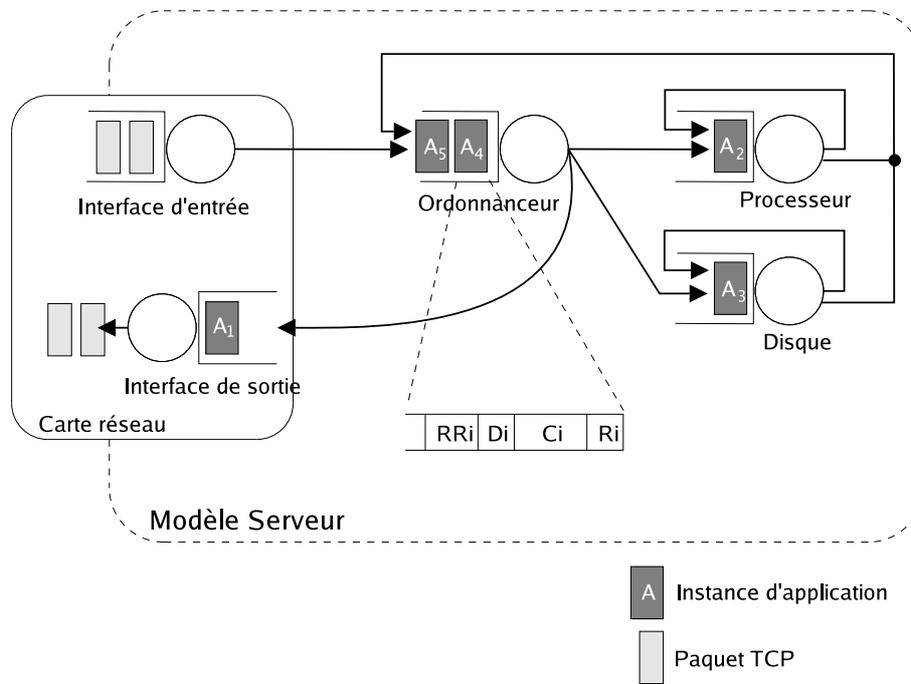


FIGURE 2.2 – Modèle serveur

### 2.2.3.3 Le système d'exploitation

Dans le cadre de ce travail, trois composants du système d'exploitation peuvent influencer les performances du système modélisé : le système d'ordonnancement des processus (ce qui revient à pouvoir simuler le mécanisme de calcul de priorité et de temps processeur alloué), le mécanisme de SWAP (modéliser dans notre cas par une probabilité d'avoir un défaut de page en faisant le rapport sur la mémoire utilisée par le processus sur la taille total de mémoire vive plus SWAP) et la gestion de l'interaction avec la carte réseau (l'impact de la gestion des interruptions étant faible ; dans notre cas le modèle proposé dans la figure 2.2 suffit).

### 2.2.3.4 Les machines

Les performances d'une machine dépendent de plusieurs facteurs : l'architecture matérielle de la machine (inter-connexion et performances des bus de communication), les performances intrinsèques de chaque composant de la machine et l'utilisation faite de ces composants par le système d'exploitation. Les machines utilisées pour constituer des grappes de calcul et des grilles possèdent des architectures matérielles et des composants semblables à ceux utilisés pour les machines grand public.

Le modèle proposé (figure 2.2), utilise des files d'attente pour représenter le processeur, le disque et les entrées/sorties réseau. La mémoire, quant à elle, est représentée par un compteur dont la valeur est bornée par la taille de la mémoire physique du serveur. Ce compteur est incrémenté ou décrémenté à chaque création ou destruction d'une instance d'application. Une file supplémentaire, appelée ordonnanceur permet d'aiguiller les requêtes de traitement vers la ressource souhaitée, en adéquation avec l'état d'avancement de l'application. Chaque file possède son propre mode de gestion des requêtes.

### 2.2.3.5 Le réseau

Notre travaux se sont appuyés sur le simulateur DHS [45]. Il permet de décrire et simuler des interconnexions de réseaux complexes. Il s'appuie à la fois sur la simulation événementielle et analytique.

## 2.2.4 Quelques résultats

Les tests ont pour objectif de valider la simulation d'une application MPI maître/esclave sur un cluster. Une attention particulière est portée à la validité de la simulation des communications réseau et des communications bloquantes de la norme MPI. Cette validation repose sur la comparaison entre les durées d'exécution réelle d'une application parallèle, sur une grappe de machines de référence, et l'estimation de ces durées obtenue par simulation. L'application parallèle utilisée est une application de test permettant de faire varier le grain de calcul.

Les figures 2.3 et 2.4 présentent les résultats obtenus pour une application moyen grain (tests 2 et 5) en fonction du réseau, de la charge et de la finesse de modélisation de la pile protocolaire.

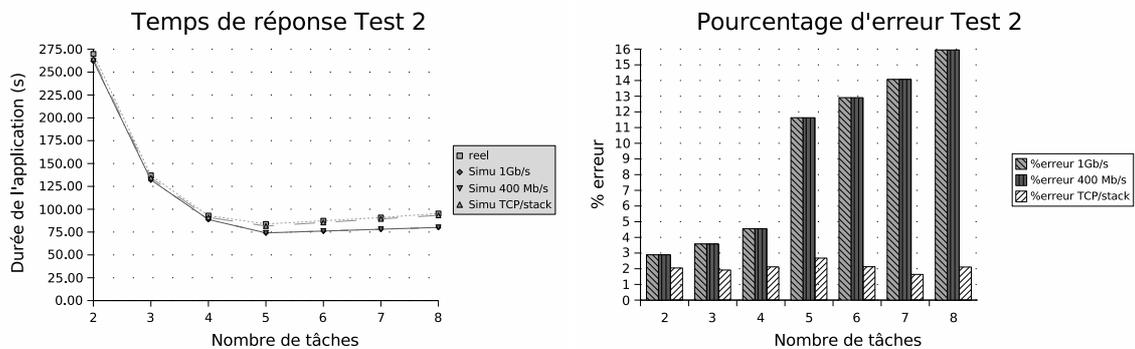


FIGURE 2.3 – Application moyen grain, esclaves lents

Les modèles développés et ajoutés au simulateur DHS ont été validés. La première étude, concernant les serveurs Web a permis d'analyser le comportement des modèles clients, applications séquentielles et machines. Cette étude a notamment permis de valider la prise en compte de la contention au niveau processeur et disque, lorsque plusieurs applications s'exécutent en concurrence sur une même machine. La prise en compte des performances intrinsèques de la machine a également été mise en évidence par la simulation de grappes de machines hétérogènes.

La simulation d'applications parallèles, et notamment la prise en compte des communications réseaux entres les applications a permis de mettre en évidence la qualité des prédictions

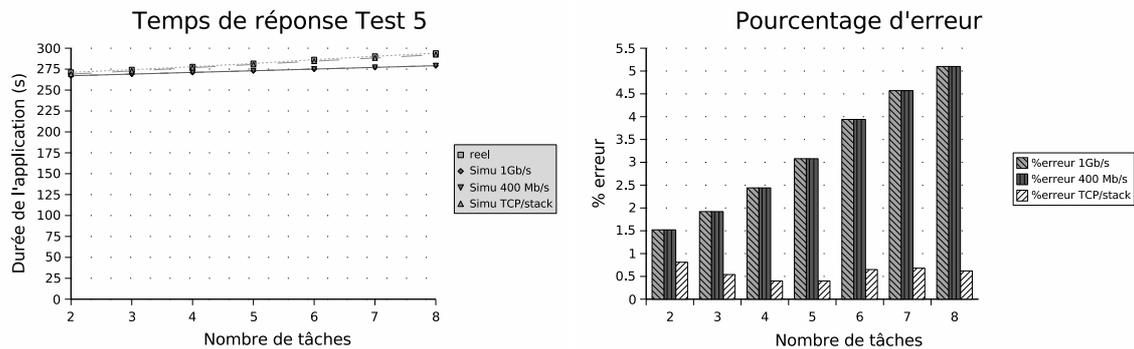


FIGURE 2.4 – Application moyen grain, maître lent

obtenues en fonction du niveau de détails du modèle de simulation utilisé et du grain de l'application [87].

#### Encadrement de thèse et publications :

- Thèse de Samuel Richard
- S.Richard, B. Miegemolle, T. Monteil, J.M. Garcia, Performance of MPI parallel applications, International Conference on Software Engineering Advances (ICSEA'2006), Papeete (France), 29 Octobre - 3 Novembre 2006, 6p.

## 2.3 Contribution sur la prédiction de charge analytique

### 2.3.1 Contexte

Nous supposons disposer d'un algorithme de placement qui cherche à minimiser le temps d'exécution des applications parallèles qui lui sont soumises. Pour cela, il explore les différentes possibilités de placement sur un ensemble de machines des différentes tâches qui lui ont été fournies en paramètre.

### 2.3.2 La problématique traitée

L'objectif est de fournir à l'algorithme de placement une estimation du temps que va mettre une tâche si elle est placée sur une machine multi-processeurs donnée. Il est supposé que le temps processeur demandé par chaque tâche de l'application est connu sur une machine de référence. On dira que l'application est déterministe. Par contre les machines peuvent déjà exécuter d'autres choses, ceci constituera une charge pré-existante. Cette dernière est stochastique. Le but est alors de prédire le temps d'exécution des processus déterministes quand ils sont introduits dans cette charge stochastique de manière analytique.

### 2.3.3 Le modèle proposé

#### 2.3.3.1 Modèle file d'attente

Tous les processus (déterministes et stochastiques) sont ordonnancés avec la politique "round-robin" avec un quantum  $Q$ . Un modèle "round-robin" sans priorité et multiserveur

est utilisé. Ce choix a été fait suite à une étude de quelques systèmes d'exploitation [77]. Les processus stochastiques sont créés selon une loi de Poisson (caractérisée par son taux d'arrivée  $\lambda$ ), ils utilisent les processeurs pendant une durée distribuée selon une loi exponentielle (caractérisée par son taux de traitement  $\mu$ ). A une date fixée, les processus déterministes sont insérés (figure 2.5).

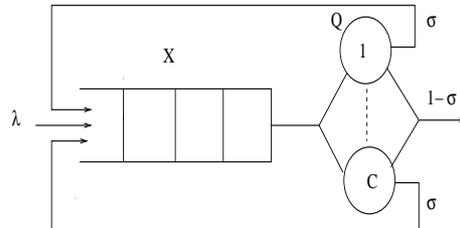


FIGURE 2.5 – Modèle “round-robin” multi-serveurs

### 2.3.3.2 Chaîne de Markov

Soit  $d$  le nombre de processus déterministes insérés simultanément sur la même machine et  $C$  le nombre de processeurs sur cette machine, le temps processeur demandé par les processus déterministes est connu, il est appelé  $t_c$  (tous les processus ou tâches de l'application ne demandent pas nécessairement le même temps). L'état du système à la date  $t$  est donné par le nombre de processus à cette date. L'état  $n$  signifie qu'il y a  $x = n$  processus dans le système. Le quantum de temps  $Q$  est petit comparé aux autres valeurs temporelles du modèle ainsi il est possible de considérer une chaîne de Markov continue. Le nouvel état du système à la date  $t + Q$  dépend uniquement de l'état à la date  $t$  et de la probabilité d'avoir un départ ou une arrivée. Ainsi, le système est Markovien.

S'il y a moins de processus que de processeurs alors tous les processus sont en exécution. S'il y a plus de processus que de processeurs alors  $C$  processus sont en exécution et  $(n - C)$  processus sont en attente. Le processus à la tête de la file s'exécutera sur le premier serveur libre. Il faut distinguer deux cas :  $d < C$  (figure 2.6) ou  $d \geq C$  car cela impacte les probabilités de transition.

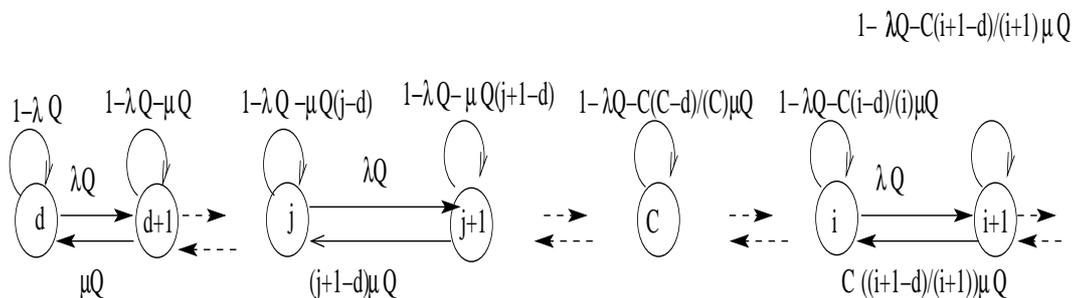


FIGURE 2.6 – Exemple de la chaîne de Markov lorsque  $d < C$

Un état stationnaire est défini comme étant un état infini où les processus déterministes sont encore dans le système. C'est comme un système dans lequel les processus déterministes

ne quitteraient jamais les files. En effet, c'est leur temps d'exécution qui est prédit donc le comportement de la machine est uniquement intéressant lorsqu'ils sont encore présents. Ainsi le nombre de processus déterministes  $d$  doit être considéré dans la condition de stabilité de la chaîne.

La chaîne de Markov est irréductible car chaque état peut être atteint à partir d'un autre. Ainsi, les probabilités limites existent toujours et sont indépendantes de la distribution initiale. Tous les états sont récurrents non-nul si après un temps infini leur état n'est pas infini. Le taux d'arrivée doit être plus petit que le taux de sortie pour avoir un état stationnaire.

### 2.3.3.3 Equation du nombre moyen de processus au cours du temps

En sommant les dérivés des probabilités de transition, on peut exprimer la variation du nombre moyen de processus dans les cas où  $d < C$  (équation 2.1 par exemple) ou  $d \geq C$  :

$$\dot{X}(t) = \lambda - C\mu(1 - P_d(t)) + \mu \sum_{i=d}^{C-1} iP_{d+C-i}(t) + C\mu d \sum_{i=C+1}^{\infty} \left(\frac{P_i(t)}{i}\right) \quad (2.1)$$

### 2.3.3.4 Approximations

Tout d'abord, une approximation de 2.2 a été proposée pour rendre l'équation 2.1 homogène et facilement intégrable.

$$\frac{\partial X_a(t)}{\partial t} = \lambda - C\mu\alpha \frac{X_a(t)}{1 + X_a(t)} + C\mu d, \quad \alpha = \frac{(\lambda + C\mu d)(1 + X^\infty)}{C\mu X^\infty} \quad (2.2)$$

Ensuite, si le nombre de processus  $x(t)$  ne varie pas beaucoup sur une machine et si sa valeur reste proche de sa moyenne. Une approximation de la date de terminaison  $t_r$  d'un processus exécuté à la date  $t_0$  et demandant  $t_c$  temps de processeur est donnée avec l'équation (2.3).

$$t_c = \int_{t_0}^{t_r} \frac{C}{X(t)} dt \quad (2.3)$$

Cette dernière peut se généraliser au cas de la prédiction du temps d'exécution de plusieurs processus déterministes insérés à la même date sur la machine.  $N$  processus triés par temps de calcul demandé :  $t_c^1 \leq t_c^2 \dots t_c^{N-1} \leq t_c^N$  sont considérés. Chaque processus quittera le système à la date  $t_r^i$  et l'équation (2.4) pourra être trouvée (avec  $t_r^0 = t_0$ ).

$$t_c^j = \sum_{i=1}^j \left( \int_{t_r^{i-1}}^{t_r^i} \frac{C}{X(t)} dt \right) \quad (2.4)$$

## 2.3.4 Quelques résultats

Des simulations de machines multiprocesseurs avec une politique round robin pour différentes charges ont permis de valider l'approximation du nombre moyen de processus et la prédiction du temps d'exécution.

A une date fixe des processus déterministes sont insérés sur des machines moyennement chargées. L'approximation de  $X(t)$  et la valeur calculée analytiquement sont comparés (cas 3 :  $d < C$  et cas 4 :  $d \geq C$ ).

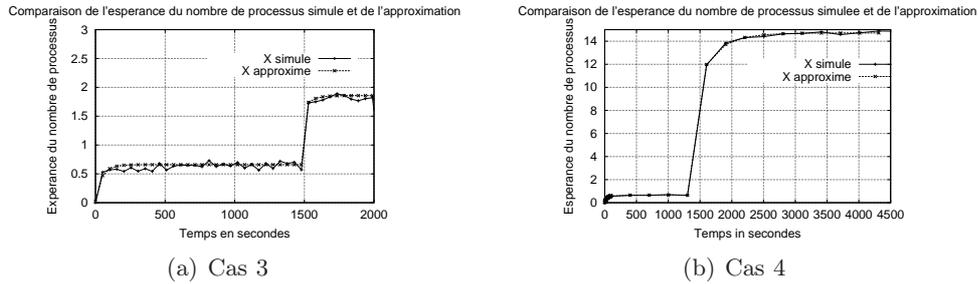


FIGURE 2.7 – Moyenne du nombre de processus dans le système

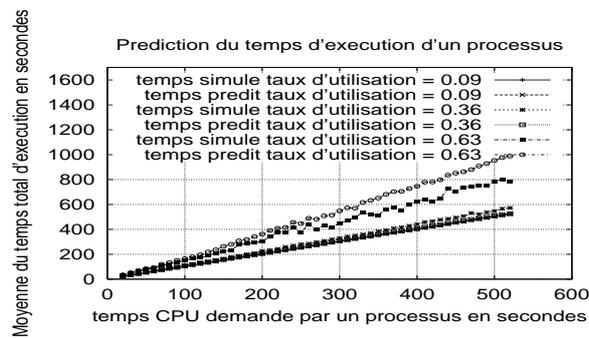


FIGURE 2.8 – Approximation du temps d'exécution d'un processus

La prédiction du temps d'exécution donne aussi de bon résultats (figure 2.8).

L'influence de la charge déterministe sur la charge stochastique a été étudiée. L'originalité de cette approche consiste à calculer les probabilités de transition influencées par l'arrivée des processus déterministes. Cette approche peut être généralisée pour différents systèmes de files d'attente. Tout d'abord, les probabilités de transition influencées par les arrivées des processus déterministes doivent être trouvées. Elles sont utilisées pour exprimer la variation de la moyenne du nombre de processus dans le système (théorie du trafic différentiel). Une approximation de cette équation peut ensuite être déduite en utilisant les propriétés en stationnaire (approximation fluide).

### Encadrement de thèse et publications :

- Thèse de Patricia Pascal
- T. Monteil, P. Pascal “Task Allocation using Processor Load Prediction on Multiprocessors Cluster” IEEE Computer Society International Workshop on Cluster Computing IWCC'01, Mangalia, Roumanie, septembre 2001.
- P. Pascal, T. Monteil, “Influence of Deterministic Customers in Time Sharing Scheduler”, Operating Systems Review, ACM, 37(1) :34-45, January 2003.

## 2.4 Contribution sur la localisation du stockage de données dans les bibliothèques de passage de messages

### 2.4.1 Contexte

Les environnements de programmation parallèle sur cluster ou grille font appel pour communiquer à deux principaux éléments de la fabrique : la mémoire pour récupérer et stocker les messages et le réseau pour transmettre le message de machines à machines. Il y a principalement deux façons de communiquer entre des tâches d'une application : de façon implicite par mémoire partagée virtuelle ou de façon explicite par passage de messages. Des outils comme PVM [46], PARMACS [20], PARFORM [22], Express [41] ou LANDA [70] ont choisi des communications par passage de messages. D'autres ont choisi un modèle de programmation par mémoire partagée : Linda [26] ou P4 [16] (sur certains types de machines).

### 2.4.2 La problématique traitée

Dans l'environnement distribué que constitue une grille ou un cluster quand une application parallèle s'exécute, il faut faire transiter des informations d'une tâche productrice à une tâche consommatrice. Dans le cas d'une application par passage de messages, on parlera d'une ou plusieurs tâches émettrices et d'une ou plusieurs tâches réceptrices. Il y a une multitude de possibilités pour réaliser le cheminement des émetteurs vers les récepteurs. Dans le logiciel LANDA dans sa version 3, les messages sont constitués de deux morceaux : une entête et les données. L'émetteur envoie l'entête vers le serveur d'entête qui est situé sur la machine de la tâche réceptrice. Un ensemble de serveurs de données est créé sur le cluster ou la grille. Ils sont à même de stocker des données provenant et à destination de n'importe quelle tâche. C'est la politique de routage qui définit la localisation des données quand un envoi de message se fait. L'objectif est de déterminer une politique optimale en fonction de l'état des machines, du réseau et des applications et de montrer que cette politique devait être dynamique. Des séries de tests ont été réalisées afin de déterminer les paramètres importants et leurs poids dans les communications entre deux tâches :

- L'hétérogénéité des machines doit être prise en compte dans la mesure où la communication avec une machine puissante est non seulement plus rapide, mais aussi moins dépendante de la charge (celle-ci étant définie par le nombre de processus en attente de processeur).
- L'influence de la charge sur la durée d'une communication est un facteur important dans la décision du routage. En effet, suivant la charge des machines communicantes, la durée de communication peut passer du simple au quadruple.
- La perturbation des calculs en cours sur le récepteur par la communication avec un autre processus ne doit pas être négligée. Les tests montrent que la communication est tellement prioritaire que les tâches en calcul, privées de CPU, ne peuvent presque plus calculer.
- L'activité de l'autre tâche doit être prise en compte. Les communications des deux tâches peuvent se gêner. Ainsi, si la décision de routage est connue pour une tâche, il semble qu'il faille éviter de router vers le même serveur pour éviter l'engorgement de ce serveur par les deux tâches.

### 2.4.3 Résolution sur un cas d'école

L'objectif est de déterminer la politique optimale de routage dans le cadre d'un système simplifié. Nous démontrons ainsi l'intérêt d'un tel mécanisme. Une généralisation de cette étude permettrait de gérer des tables de routage dans le système LANDA. Dans la suite, nous introduisons un modèle d'un système particulier qui nous permet, sous des hypothèses simplificatrices, d'étudier ce problème dans le cas d'algorithmes synchrones. Ce modèle est basé sur la théorie des processus de décision markoviens [12]. Cette étude peut se rapprocher des problèmes de gestion de fichiers et du choix de localisation des serveurs de fichiers sur un réseau de stations de travail [78]. De même, nous pouvons trouver certaines analogies avec des problèmes de routage dans les réseaux téléphoniques [44] ou les réseaux locaux [4].

#### 2.4.3.1 Système considéré

Pour simplifier, nous considérons un système uniquement constitué de deux tâches et de trois machines. Ce système est le plus simple qui nous permette d'étudier le problème du routage. Les trois machines sont reliées par un lien de communication. Sur chacune se trouve un serveur de données. Le serveur d'en-tête n'est pas pris en compte. Nous notons :

- $T_1$  et  $T_2$ , les deux tâches se trouvant respectivement sur les machines  $M_1$  et  $M_2$
- $DS_1$ ,  $DS_2$  et  $DS_3$  les trois serveurs de données placés sur les machines  $M_1$ ,  $M_2$  et  $M_3$ .

#### 2.4.3.2 Hypothèses

Les hypothèses suivantes sont posées :

- **Modèle de programmation** :  $T_1$  et  $T_2$  sont identiques, c'est à dire qu'elles effectuent les mêmes quantités de calcul et de communication. Cette hypothèse peut être vue comme celle d'un mode de programmation SPMD (Simple Program Multiple Data). Cela ne signifie pas pour autant que les deux tâches soient identiques du point de vue de l'observation. En effet, l'influence de l'environnement externe (les autres utilisateurs) induit des comportements différents pour les deux tâches. Par exemple, les tâches effectuent plus ou moins vite leur calcul suivant la charge de la machine sur laquelle elles s'exécutent.
- **Grain de l'algorithme** : Dans ce système, une tâche alterne entre des phases de calcul et de communication. Les algorithmes parallèles auxquels est destinée la machine LANDA sont de type gros grain. Le temps passé par  $T_1$  et  $T_2$  en phase de calcul est grand devant le temps consacré aux communications. Nous supposons que ce temps de calcul est une variable aléatoire de moyenne  $1/\mu$ . Il correspond au temps passé en calcul par les tâches entre deux communications. Il ne tient pas compte de l'influence de la charge.
- **Taille des messages** : La taille des messages est une variable aléatoire de moyenne  $M$ . En corrélation avec  $\mu$ , le paramètre  $M$  permet de représenter le grain de l'algorithme, nous pouvons alors déduire le temps passé en communication entre deux phases de calcul à partir du débit offert par le réseau.  $M$  nous permet d'étudier la politique optimale de routage en fonction de la taille moyenne des messages.
- **Communications bloquantes et durée des communications** : Les communications sont de deux sortes : écritures et lectures. Dans les deux cas, elles sont construites à partir du mécanisme des RPC (Remote Procedure Call) et sont donc bloquantes. Lors

d'une écriture, une tâche envoie des données à un serveur de données où une autre tâche pourra venir les lire. Lors d'une lecture, une tâche vient lire les données dans le serveur où elles ont été préalablement stockées. La durée d'une écriture distante (ou d'une lecture distante) d'une tâche dépend de la taille du message, du débit réseau  $1/\tau$ , de la charge des machines impliquées dans la communication, ainsi que de l'activité de l'autre tâche qui peut perturber l'écriture (ou la lecture) par sa propre activité de communication. Lorsque l'écriture (ou la lecture) est locale, seuls rentrent en compte la taille du message, la charge de la machine et l'activité de l'autre tâche. Dans tous les cas, la durée d'une communication est petite devant le temps d'une phase de calcul.

- **Synchronisme** : Nous nous restreignons à des algorithmes synchrones. Le schéma de fonctionnement des tâches T1 et T2 de l'algorithme est alors le suivant. Une tâche calcule, puis passe en écriture. Lorsqu'elle a terminé son écriture, elle se retrouve dans un état de synchronisation, en attente du message que doit lui envoyer l'autre tâche. Lorsque ce message est disponible, elle le lit. C'est seulement une fois que cette lecture est terminée, qu'elle peut repasser en calcul.
- **Charge des machines** : Pour prendre en compte l'activité des utilisateurs qui travaillent sur les stations composant la machine LANDA, nous considérons leur charge. L'indicateur est le nombre de processus en exécution supposé constant. Il y a deux motivations à ce choix. Tout d'abord, ce critère permet des mesures simples et aisément reproductibles. Ensuite, il facilite la confrontation des résultats obtenus avec la réalité. Les valeurs de ces variables sont respectivement  $P_c^1$ ,  $P_c^2$  et  $P_c^3$  pour les machines  $M_1$ ,  $M_2$  et  $M_3$ . En faisant varier ces paramètres de charge, nous pouvons déterminer le routage le mieux approprié dans une configuration de charge donnée.

L'état du système est entièrement déterminé par la donnée des paramètres  $\mu$ ,  $M$ ,  $P_c^1$ ,  $P_c^2$ ,  $P_c^3$ ,  $\tau$ , l'activité de chacune des tâches et par l'état des différentes boîtes aux lettres.

### 2.4.3.3 Modèle

Le système est représenté sous forme d'une chaîne de Markov. Telle qu'elle est définie, elle comporte une centaine d'états. L'espace d'état du système doit être partitionner en classes afin de réduire la complexité. Une fois les probabilités de transition entre ces classes d'états définies, les probabilités de transition entre les états d'une même classe sont déterminées (i.e au niveau des états des boîtes aux lettres). Ces changements sont supposés immédiats lorsque l'action les ayant provoqués est terminée [68].

Une politique n'est optimale que vis à vis d'un certain critère de performance, plusieurs sont possibles :

- **Temps passé en communication** : Il est souvent retenu en parallélisme. Mais dans notre cas, il ne permet pas de prendre en compte les perturbations induites par la communication sur le calcul en cours sur le récepteur. Ainsi, la minimisation globale du temps passé en communication peut très bien se faire aux dépens des calculs.
- **Temps passé en état de synchronisation** : Il semble a priori intéressant dans la mesure où il cherche à minimiser le temps d'attente des tâches. Il tend à faire en sorte que les deux tâches progressent au même rythme. Pour autant, rien ne garantit que la durée de l'application s'en trouvera réduite. En effet, des décisions de routage ralentissant volontairement les tâches pour minimiser le temps d'attente en état de synchronisation peuvent être prises.

- **Puissance de calcul** : Il exige des tâches qu'elles passent le plus de temps possible en calcul. Pour cela, elles ne doivent pas perdre de temps en communication. Ce critère assure également un routage qui minimise le déséquilibre entre les tâches (il favorise celle qui est la plus en retard, i.e. sur la machine la plus chargée). Enfin, ce critère permet de prendre en compte les perturbations des calculs induites par les communications. C'est ce dernier qui a été retenu.

Le problème du routage peut être vu comme un problème de contrôle d'un processus Markovien ayant un espace d'état fini et un ensemble de commandes fini et non vide dans chaque état. Le processus Markovien créé est ergodique. En effet, en partant d'un état initial quelconque, en un nombre fini d'étapes, nous pouvons atteindre un état de référence bien choisi. Une méthode de résolution par approximation successive a été choisie.

#### 2.4.4 Quelques résultats

Dans le cas où  $M_1$  et  $M_2$  sont toutes les deux peu chargées, la politique optimale de routage est identique au cas homogène de charge et correspond à un routage à la source pour les deux tâches, c'est à dire que les tâches stockent les données sur le serveur de données situé sur leur machine afin de ne pas perturber la tâche réceptrice. Par contre, dans les situations où il y a un fort déséquilibre de charge entre  $M_1$  et  $M_2$ , la politique optimale de charge propose un routage sur la machine  $M_3$  (celle qui ne possède pas de tâche) pour la tâche la plus chargée et un routage à la source pour la tâche la moins chargée (figure 2.9, politique de routage 5 et 6).

numéro de politique

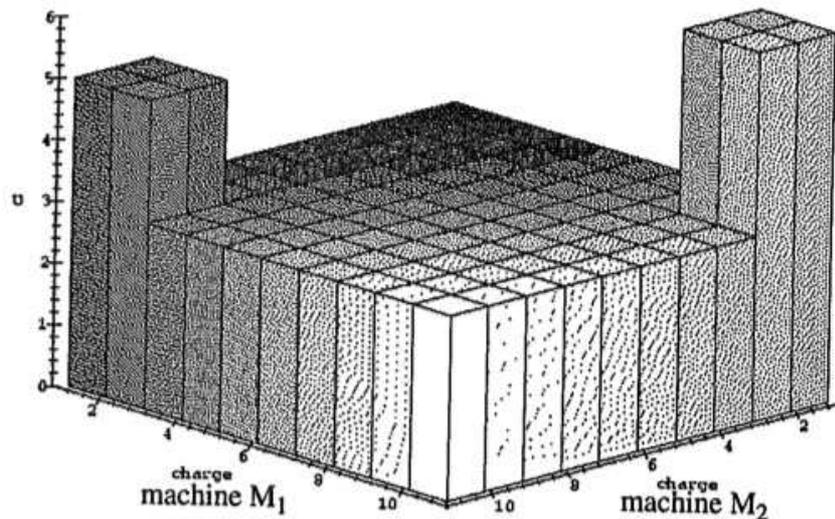


FIGURE 2.9 – choix du routage en fonction de la charge

L'interprétation des résultats est délicate. Tout d'abord parce que l'optimisation du routage est globale, et donc qu'il n'est pas vraiment possible de raisonner sur une tâche à la fois. Ensuite

parce que de nombreux phénomènes jouant en sens contraire doivent être pris en considération. Nous pouvons tout de même déjà constater que :

- le routage sur une troisième machine peut être valable dans certaines configurations de charge lorsque une machine puissante est disponible. Par contre, si celle-ci est trop chargée, il perd tout intérêt.
- Le routage à la source se révèle la plupart du temps le plus intéressant. Les tâches évitent de se gêner lors de phase de calcul. Elle préfèrent se retarder un peu lors de la lecture en réalisant une communication croisée.

La version 3.0 de LANDA a permis de tester différents routages et peut mettre en œuvre n'importe quel type de routage. Néanmoins, la fin des développements sur LANDA n'a pas permis de travailler sur la généralisation de ces principes à un grand nombre de tâches avec des schémas de communication diverses.

#### **Encadrement de thèse et publications :**

- Thèse de Thierry Monteil
- Thèse de David Gauchard
- T. Monteil, J.M Garcia, D. Gauchard, O. Brun, "Communication Kernel for High Speed Networks in the Parallel Environment LANDA-HSN". IEEE Computer Society International Workshop on Cluster Computing, conférence IWCC'99, Melbourne, décembre 1999.

## **2.5 Contribution sur la reconfiguration du réseau pour minimiser la consommation énergétique sous des contraintes de QoS**

### **2.5.1 Contexte**

Ces dernières années, une nouvelle problématique est née au vu des considérations écologiques de plus en plus pressentes. Elle concerne une gestion plus performante de l'énergie pour les machines et Internet plus généralement. Les centres de ressources informatiques consomment 0.5% de l'électricité mondiale et produisent 0.2% des émissions de carbone soit presque l'équivalent de ce que rejette un pays comme l'Argentine ou les Pays Bas. Si on ne fait rien, dans 25 ans, Internet consommera autant d'énergie que l'humanité tout entière en 2008 [50, 32]. De nombreux travaux existent sur la gestion énergétique des machines et l'accès au réseau mais peu sur l'utilisation du cœur de réseau par les applications communicantes et le couplage à la qualité de service (QoS) [52].

Les caractéristiques de QoS sont des valeurs mesurables. Ces dernières peuvent être classées selon différentes familles :

- **caractéristiques temporelles** : délai de transit, délai de réponse, délai d'établissement, gigue
- **caractéristiques de capacité** : cadence d'entrée, débit et cadence de sortie, cadence des données utilisateur, cadence des données de contrôle
- **caractéristiques d'intégrité** : probabilité de panne de connexion, probabilité de perte
- **caractéristiques de sécurité** : protection de la connexion, authentification

Dans la suite, nous définissons un profil de QoS comme un ensemble de valeurs caractérisant les besoins en QoS pour le bon fonctionnement d'une application. Nous supposons que les ap-

plications considérées disposent de plusieurs profils de QoS et qu'il est possible de reconfigurer une application pour la faire basculer d'un profil à un autre.

## 2.5.2 La problématique traitée

Nous nous plaçons dans le cadre d'un centre de ressources informatiques constitué de nombreuses machines regroupées en différents clusters et interconnectées par un maillage de routeurs permettant des redondances afin d'amener plus de robustesse et de capacité à absorber le trafic fluctuant généré. Un ensemble d'applications distribuées est placé sur ces différentes machines. Ces dernières communiquent en utilisant le cœur de réseau. Elles ont aussi la capacité de pouvoir fonctionner avec différents profils. On peut prendre par exemple un service de fourniture de vidéo avec différentes qualités d'image et donc de besoins en débit ou cadence de sortie. L'objectif de l'étude est d'offrir la meilleure qualité possible en minimisant la puissance consommée par les équipements réseaux. Afin de laisser le choix entre qualité de service et consommation, nous mettrons en place un critère d'optimisation intégrant les deux aspects. Au niveau du réseau, nous disposons de gros routeurs constitués de plusieurs cartes offrant chacune plusieurs connexions. Nous pouvons gérer l'allumage : des ports d'un module, d'un module ou du châssis dans lequel s'insère les modules. Afin de simplifier un peu le problème au niveau des applications nous supposerons qu'une application est constituée d'un émetteur sur une machine et d'un récepteur sur une autre machine.

## 2.5.3 Le modèle proposé

### 2.5.3.1 Variables et fonctions pour la partie réseau

La topologie du réseau est représentée sous forme d'un graphe orienté  $G = \{N, E\}$  où  $\mathbf{N}$  est l'ensemble des noeuds et  $\mathbf{E}$  est l'ensemble des arrêtes. Un couple  $(i, j)$  représente le lien réseau entre le noeud  $i$  et le noeud  $j$ . On pose :

- $|\mathbf{N}| = n_N$  et  $|\mathbf{E}| = n_E$
  - $c_{i,j}^e$  la capacité de l'arc  $(i, j)$
  - $p_{i,j}^e$  la puissance électrique utilisée par le lien de  $i$  vers  $j$ , c'est à dire la puissance électrique consommée par le port réseau du châssis  $i$  uniquement
  - $z_{i,j}^e$  qui définit si le port partant de  $i$  vers  $j$  est allumé électriquement,  $z_{i,j}^e$  vaut 1 si le port est allumé, 0 sinon
  - $p_{i,l}^m$  la puissance électrique consommée par le module  $l$  du châssis  $i$
  - $z_{i,l}^m$  définit si le module  $l$  du châssis  $i$  est allumé
  - $p_i^c$  la puissance électrique utilisée par le châssis  $i$
  - $z_i^c$  définit si le châssis  $i$  est allumé électriquement
- Soit l'ensemble des modules pour un châssis  $i$  :

$$\mathbf{M}_i = \{m_1, \dots, m_l, \dots, m_{n_{M_i}}\}, |\mathbf{M}_i| = n_{M_i}$$

La fonction NodeModules donne l'ensemble des modules pour un châssis.

$$\text{NodeModules} : \mathbf{N} \rightarrow \mathbf{M}$$

$$\forall i \in \mathbf{N} : \mathbf{M}_i = \text{NodeModules}(i)$$

La fonction `ModuleCards` donne l'ensemble des ports (ou l'ensemble des arcs sortants) pour un module.

$$\text{ModuleCards} : \mathbf{M} \rightarrow \mathbf{E}$$

Soit l'ensemble des arcs  $\mathbf{E}_{i,l}$  sortant des ports du module  $l$  du chassis  $i$ .

$$\forall i \in \mathbf{N}, \forall l \in [1, n_{M_i}] : \mathbf{E}_{i,l} = \text{ModuleCards}(M_i)$$

L'ensemble  $\mathbf{E}_i$  des ports d'un chassis  $i$  peut donc s'écrire comme l'union de tous les ports de tous les modules du chassis :

$$\forall i \in \mathbf{N} : \mathbf{E}_i = \bigcup_{l \in [1, n_{M_i}]} \mathbf{E}_{i,l}$$

### 2.5.3.2 Relations pour la partie réseau

La puissance consommée par le réseau s'écrit comme la somme pour tous les routeurs de la consommation du chassis de ses modules et de ses ports :

$$P_{total} = \sum_{i \in \mathbf{N}} \{ p_i^c \cdot z_i^c + \sum_{l \in [1, n_{M_i}]} [ p_{i,l}^m \cdot z_{i,l}^m + \sum_{j \in \mathbf{E}_{i,l}} p_{i,j}^e \cdot z_{i,j}^e ] \}$$

En regroupant par famille de composant :

$$P_{total} = \sum_{i \in \mathbf{N}} p_i^c \cdot z_i^c + \sum_{i \in \mathbf{N}, l \in [1, n_{M_i}]} p_{i,l}^m \cdot z_{i,l}^m + \sum_{i \in \mathbf{N}, j \in \mathbf{E}_i} p_{i,j}^e \cdot z_{i,j}^e$$

Il existe ensuite un ensemble de contraintes sur l'allumage ou non des différents éléments constituant le routeur. Si tous les ports d'un module sont éteints, le module peut être éteint. Ceci se traduit par l'implication suivante :

$$\forall i \in \mathbf{N}, \forall l \in [1, n_{M_i}] : \sum_{(i,j) \in E_{i,l}} z_{i,j}^e = 0 \Rightarrow z_{i,l}^m = 0$$

Afin de conserver un critère linéaire on peut écrire cette implication comme la conséquence d'un ensemble d'inégalités :

#### Contrainte 1

$$\forall i \in \mathbf{N}, \forall l \in [1, n_{M_i}] : z_{i,l}^m - \sum_{j \in \mathbf{E}_{i,l}} z_{i,j}^e \leq 0$$

#### Contrainte 2

$$\forall i \in \mathbf{N}, \forall l \in [1, n_{M_i}], j \in \mathbf{E}_{i,l} : z_{i,j}^e - z_{i,l}^m \leq 0$$

De même, si tous les modules d'un chassis sont éteints, le chassis peut être éteint. Ceci s'écrira sous forme linéaire :

#### Contrainte 3

$$\forall i \in \mathbf{N} : z_i^c - \sum_{l \in [1, n_{M_i}]} z_{i,l}^m \leq 0$$

#### Contrainte 4

$$\forall i \in \mathbf{N}, l \in [1, n_{M_i}] : z_{i,l}^m - z_i^c \leq 0$$

Un lien éteint dans un sens doit être éteint dans l'autre :

#### Contrainte 5

$$\forall i, j \in \mathbf{N} : z_{i,j}^e - z_{j,i}^e = 0$$

### 2.5.3.3 Variables et fonctions pour la partie application

Soient :

- $\mathbf{A}$  l'ensemble des applications utilisant le réseau :  $\mathbf{A} = \{a_1, \dots, a_k, \dots, a_{n_A}\}$
- $|\mathbf{A}| = n_A$
- $a_k^s$  représente la partie émettrice de l'application  $a_k$
- $a_k^r$  représente la partie réceptrice de l'application  $a_k$ . On suppose que les communications sont unidirectionnelles (émetteur vers récepteur) dans l'application et on néglige tous les trafics retours (signalisation, retour, etc)
- $N_s^k$  représente le chassis (routeur de bordure) où se trouve connectée la machine de l'émetteur de l'application  $a_k$
- $N_r^k$  représente le chassis (routeur de bordure) où se trouve connectée la machine du récepteur de l'application  $a_k$
- l'ensemble  $\mathbf{Q}^k$  représente tous les profils en QoS possibles pour l'application  $a_k$ . Chaque élément de cet ensemble est lui-même constitué d'un ensemble de valeurs exprimant les caractéristiques élémentaire de QoS demandées par l'application (débit, gigue, temps de réponse, etc) qui sont regroupés dans un tuple. Ce dernier est constitué de valeurs, d'intervalle ou de toutes autres représentations permettant de caractériser un besoin élémentaire en terme de qualité de service.

$$\mathbf{Q}^k = \{q_1^k, \dots, q_s^k, \dots, q_{n_{Q^k}}^k\}, |\mathbf{Q}^k| = n_{Q^k}$$

- La fonction  $M$  représente un métrique permettant de mesurer la qualité d'un profil. Cette dernière permet de définir une relation d'ordre total notée  $<$  entre les différents profils d'une application. Afin de simplifier par la suite la notation, on supposera que les besoins sont rangés suivant les indices croissants dans l'ensemble  $\mathbf{Q}^k$  selon l'ordre  $<$  défini avec la métrique  $M$  (ceci est toujours réalisable à une permutation prêt sur les indices) :

$$M(q_1^k) < M(q_2^k) < \dots < M(q_{n_{Q^k}}^k)$$

- $\mathbf{B}^k = \{b_1^k, \dots, b_n^k, \dots, b_{n_{B^k}}^k\}$  représente l'activation ou non des profils possibles pour une application  $a_k$ . Si  $b_n^k$  est égal à 1, c'est le profil  $n$  qui est actif pour l'application  $k$  et dans ce cas tous les autres variables représentant l'activation des profils pour l'application  $k$  sont à 0.
- $|\mathbf{B}^k| = n_{B^k}$
- la fonction  $AF^k$  donne pour une application  $a_k$  et un profil choisi  $q_n^k$  le débit moyen produit par l'application. Notre supposons donc que le débit moyen produit par l'application est fonction du profil choisi.
- $x_{i,j}^k$  est le débit utilisé sur le lien  $(i, j)$  par l'application  $a_k$ .

- La fonction RoutingNodes pour le réseau prend un chassis de départ et un chassis d'arrivée et crée l'ensemble des chassis de tous les chemins possibles calculés par la fonction de routage du réseau (OSPF, RIP, RIPv2 ...). Cette fonction prend en compte l'extinction et l'allumage des éléments du réseau.
- pour l'application  $a_k$ ,  $RN^k$  correspondant aux chassis calculés tels que :

$$RN^k = \text{RoutingNodes}(N_s^k, N_r^k)$$

- On définit aussi, pour un noeud  $i$ , les ensembles  $\mathbf{RN}_{\text{input}}^{k,i}$  et  $\mathbf{RN}_{\text{output}}^{k,i}$  suivants :

$$\mathbf{RN}_{\text{input}}^{k,i} : \forall j \in \mathbf{N}, j \in RN^k, \exists e_{i,j} \in \mathbf{E}$$

$$\mathbf{RN}_{\text{output}}^{k,i} : \forall j \in \mathbf{N}, j \in RN^k, \exists e_{j,i} \in \mathbf{E}$$

#### 2.5.3.4 Relations pour la partie application

Pour une application, un seul profil peut être actif à un instant donné, ce qui s'écrit par la contrainte suivante :

##### Contrainte 6

$$\forall k \in [1, n_A] : \sum_{n=1}^{n_{B^k}} b_n^k - 1 = 0$$

il faut respecter les capacités des liens,  $z_{i,j}^e$  et tenir compte des liens éteints :

##### Contrainte 7

$$\forall (i, j) \in \mathbf{E} : \sum_{k=1}^{n_A} (x_{i,j}^k - c_{i,j}^e \cdot z_{i,j}^e) \leq 0$$

Les flots se propagent de  $a_k^s$  à  $a_k^r$ . Pour tous les chassis, la différence entre les débits entrants et sortants est nulle :

##### Contrainte 8

$$\forall k \in [1, n_A], \forall i \in \mathbf{RN}^k : \sum_{j \in \mathbf{RN}^k} x_{i,j}^k - \sum_{u \in \mathbf{RN}^k} x_{u,i}^k = 0$$

Le débit moyen généré par une application sort du routeur de bordure  $N_s^k$  :

##### Contrainte 9

$$\forall k \in [1, n_A] : \sum_{i \in \mathbf{RN}^k} x_{N_s^k, i}^k - \sum_{j=1}^{n_{B^k}} (AF^k(q_j^k) \cdot b_j^k) = 0$$

Le débit moyen généré par une application entre dans le routeur de bordure  $N_r^k$  :

##### Contrainte 10

$$\forall k \in [1, n_A] : \sum_{i \in \mathbf{RN}^k} x_{i, N_r^k}^k - \sum_{j=1}^{n_{B^k}} (AF^k(q_j^k) \cdot b_j^k) = 0$$

Une perte de qualité de service (Quality Loss)  $QL$  est défini ainsi :

$$\forall k \in [1, n_A] : QL_{a_k} = M(q_{n_{B^k}}^k) - \sum_{j=1}^{n_{B^k}} b_j^k \cdot M(q_j^k)$$

La perte de qualité de service totale pour le réseau s'écrit donc :

$$QL_{Total} = \sum_{a_k \in \mathbf{A}} QL_{a_k}$$

### 2.5.3.5 Formalisation du problème

De manière globale, le problème peut s'écrire comme la minimisation d'un critère, en parcourant les possibilités pour les différentes variables du problème :

$$\begin{aligned} \text{Min}_{z_i^c, z_{i,l}^m, z_{i,j}^e, b^k} \alpha \cdot \{ & \sum_{i \in \mathbf{N}} p_i^c \cdot z_i^c + \sum_{i \in \mathbf{N}, l \in [1, n_{M_i}]} p_{i,l}^m \cdot z_{i,l}^m + \sum_{i \in \mathbf{N}, j \in \mathbf{E}_i} p_{i,j}^e \cdot z_{i,j}^e \} \\ + \beta \cdot (1 - \alpha) \cdot \sum_{a_k \in \mathbf{A}} \{ & M(q_{n_{B^k}}^k) - \sum_{j \in [1, n_{B^k}]} b_j^k \cdot M(q_j^k) \} \end{aligned}$$

La valeur de  $\alpha$  (comprise entre 0 et 1) est constante et représente le poids relatif entre la satisfaction en terme de qualité de service et l'économie d'énergie. Le coefficient  $\beta$  est un coefficient de normalisation permettant de rendre comparable qualité de service et économie d'énergie. Il se calcule ainsi :

$$\beta = \frac{p_{min} + p_{max}}{d_{min} + d_{max}}$$

avec  $p_{min}$  puissance minimale (calcul du critère avec  $\alpha = 1$ ),  $p_{max}$  puissance maximale (tout allumé),  $d_{min}$  débit minimal injectable (tous les profils au minimum) et  $d_{max}$  débit maximum injectable (calcul du critère avec  $\alpha = 0$ ).

### 2.5.3.6 Le routage

La formalisation précédente permet d'obtenir des solutions ne prenant pas en compte des contraintes de routage particulier. Ceci permet d'obtenir une borne minimum sur le critère. Toutefois dans la réalité, le routage suit des règles. Nous nous sommes intéressés plus particulièrement au routage OSPF avec équilibrage de charge avec une métrique constante sur les liens (par exemple le débit maximum).

Les flots respectent les contraintes de load balancing en fonction du protocole de routage choisi. Pour OSPF ces contraintes définissent une division des flots équitable entre les chemins de même coût :

#### Contrainte 11

$$\forall k \in [1, n_A]; \forall i \in \mathbf{RN}^k, \text{Card}(\mathbf{RN}_{\text{output}}^{k,i}) > 1, i \neq N_s^k; \forall j \in \mathbf{RN}_{\text{output}}^{k,i} :$$

$$x_{i,j}^k = \frac{1}{\text{Card}(\mathbf{RN}_{\text{output}}^{k,i})} \sum_{l \in \mathbf{RN}_{\text{input}}^{k,i}} x_{l,i}^k$$

Notre problème se complique car les messages émis d'un émetteur vers un récepteur vont aussi maintenant être contraint par OSPF qui va autoriser uniquement les chemins minimisant sa métrique. Le problème devient "un minimum de minimum". Nous avons choisi de résoudre cela en deux étapes une première recherche à éteindre des équipements et fixe le routage, une seconde recherche les profils minimum. Pour la première étape nous avons mis au point une heuristique cherchant à couper d'abord les chassis faisant transiter le moins d'applications puis les modules selon le même critère de liste puis enfin les ports.

#### 2.5.4 Quelques résultats

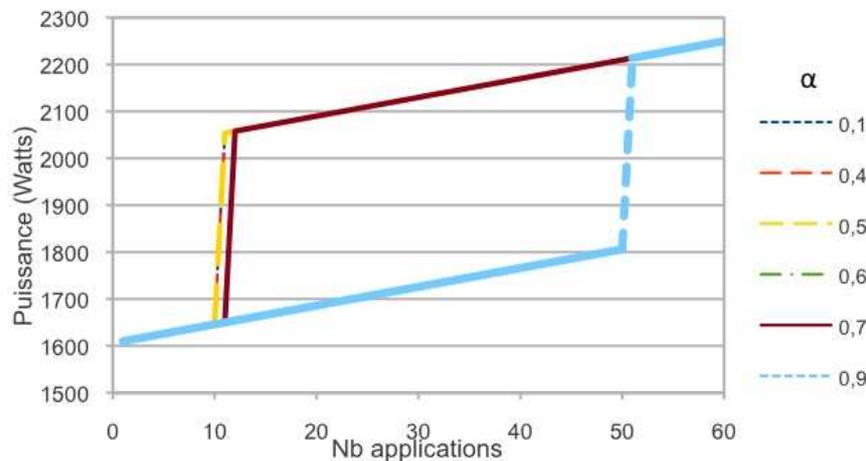


FIGURE 2.10 – Puissance consommée en fonction du nombre d'applications et de  $\alpha$

La topologie de la grille grid'mip a été choisie comme test [30]. Le logiciel Cplex [29] a été utilisé. Nous mettons de une à cent applications. Chaque application est constituée d'un émetteur et d'un récepteur. Tous les émetteurs sont placés sur le même cluster et sont donc reliés au même routeur de bordure, de même pour les récepteurs. Les applications disposent de cinq profils caractérisés par des besoins en débit de : 200Mo/s, 400Mo/s, 600Mo/s, 800Mo/s et 1Go/s. Sur la figure 2.10, le nombre d'application placées sur la grille augmente progressivement. Le saut en énergie correspond à l'allumage d'un nouveau routeur dans le cas d'OSPF afin de créer un chemin supplémentaire pour écouler le trafic. En fonction de  $\alpha$  et donc du poids que l'on donne à la dégradation de la qualité de service, cet allumage intervient plus ou moins tôt.

#### Encadrement de thèse et publications :

- Thèse de Rémi Sharrock
- R. Sharrock, P. Stolf, T. Monteil, Extending TUNE for Autonomous Management of QoS at Application and Network Levels, International Conference on Networks (ICN 2010), Ménuires (France), 11-16 Avril 2010, pp.327-330

## 2.6 Bilan

Dans ce chapitre, différentes techniques ont été manipulées : simulation événementielle, simulation analytique, système à base de files d'attente, processus de décision Markovien, chaîne de Markov, optimisation en variables entières de systèmes linéaires contraints.

Ceci a permis de fournir des modèles plus fins de plate-forme distribuée. Ces résultats ont été exploités par la société QoS Design et devrait se poursuivre sur de futurs projets de recherche dans le domaine du "cloud" et de l'accès à des services distants.

Le travail sur la localisation des messages a montré que les environnements parallèles de communication par passage de messages pourrait encore accélérer leur performance en mettant en place des politiques de routage dynamique des messages. Toutefois, la généralisation des premiers résultats que nous avons obtenu n'est pas triviale et il n'est pas sûr que le gain obtenu ne soit pas perdu par la lourdeur des mécanismes à mettre en place.

Enfin la dernière contribution s'attaque au dilemme performance / consommation d'énergie pour la partie réseau. Les premiers résultats sont intéressants et ouvrent la porte à de nouvelles collaborations. Ceci constitue un challenge pour les années à venir car le système parfait doit pouvoir tenir compte de la consommation énergétique des machines et du réseau, de la qualité de service et du coût financier par exemple. Ce travail va donc se poursuivre en le couplant plus fortement à des politiques de gestion autonome.

# Chapitre 3

## La couche service

### 3.1 Introduction

La couche service va offrir un ensemble de fonctionnalités qui seront utilisées par l'intergiciel. Cette couche permet une certaine abstraction des différences qu'il peut y avoir dans les ressources offertes par la couche fabrique. Nous nous intéresserons dans la suite à deux services plus particulièrement : le service d'observation de l'état des ressources et le service de placement sur les ressources avec ses algorithmes.

Trois aspects du placement seront abordés : comment intégrer des notions de classes de services dans l'ordonnanceur, comment tenir compte d'aspect économique lors d'un placement et enfin comment prendre en compte le réseau pour des placements sur une grille.

Les algorithmes de placement peuvent fonctionner en aveugle mais pour les rendre plus efficaces, ils s'appuient en général sur des services d'observation leur permettant de connaître l'état des ressources. Dans un deuxième temps, nous présenterons succinctement le logiciel Aroma et son architecture. Ce dernier a servi de cadre pour la mise en place de service d'observation et l'implémentation d'algorithmes de placement.

### 3.2 Contributions sur l'ordonnancement

#### 3.2.1 Les algorithmes

Baumgartner et Wah ont effectué une étude des algorithmes d'ordonnancement dans [9]. Un tel problème se présente avec cinq composants : les événements, l'environnement, les besoins, l'ordonnanceur et l'ordonnancement. Les caractéristiques du travail (temps de calcul, contraintes de précédence), l'environnement machine (nombre de processeurs, interconnexion, puissance des processeurs) et les objectifs de performance sont les entrées de l'ordonnanceur et l'ordonnancement est la sortie.

Il y a plusieurs niveaux d'ordonnancement dans les systèmes distribués : "intracomputer scheduling" (ordonnancement local à l'ordinateur) et "intercomputer scheduling" (ordonnancement global à travers plusieurs ordinateurs qui entraîne donc des communications). Pour chacun d'eux, l'ordonnancement est une mise en adéquation entre les événements et les ressources.

Feitelson a étudié dans [39] le placement des applications parallèles. Chaque travail paral-

lèle est exécuté dans une partition de processeurs. Cette partition peut être fixe (définie par l'administrateur système), variable (la taille de la partition est déterminée lors de la soumission basée sur la demande de l'utilisateur), adaptative (la taille de la partition est déterminée par l'ordonnanceur en fonction de la charge et de la demande de l'utilisateur), dynamique (la taille de la partition peut varier au cours de l'exécution en fonction des besoins et de la charge). Plusieurs niveaux de préemption peuvent être considérés : aucune préemption, préemption locale (la suite de l'exécution aura lieu sur les mêmes processeurs), préemption avec migration (la suite de l'exécution peut se faire sur un autre processeur), "gang scheduling" (tous les threads d'un processus sont suspendus et terminés simultanément ; il peut être implémenté avec ou sans migration). Il existe de nombreuses politiques de placement, Gibbons propose une politique d'ordonnement du type LEWF ("Least Expected Work First") qui réduit le temps de réponse par rapport à une politique FIFO, si les temps de service sont inconnus ou seulement estimés l'ordonnement de type "backfilling" donne de meilleurs résultats (elle consiste dans le cadre d'une politique FIFO à permettre à une application de s'exécuter avant son tour si cela ne retarde pas les applications avant elle). Différentes possibilités pour le placement de processus et la migration sont décrites dans [11] [89] [28].

Trois algorithmes de placement sont proposés par la suite en prenant en compte des classes de qualité de service, le réseau ou bien des aspects économiques.

## 3.2.2 Ordonnement et qualité de service

### 3.2.2.1 Contexte

La puissance des machines actuelles, le partage toujours plus grand de ces ressources pour des raisons de coût principalement que ce soit sur une grille ou un nuage rendent de plus en plus possible la cohabitation sur une même machine de plusieurs familles d'applications appartenant à plusieurs utilisateurs avec des besoins différents. Ces derniers se traduisent en terme de contraintes qui peuvent être diverses : date butoir de terminaison, date de début imposée, nécessité d'être seul sur la machine lors de l'exécution, contrainte de logiciel, niveau de priorité, etc. En fait, tout ceci définit des niveaux de qualité de service. Ceci se traduit naturellement dans la mise en œuvre de classes de services comme cela se fait classiquement dans le monde du réseau. Ce problème est encore très peu traité dans la littérature et pose des problèmes non seulement en terme d'algorithme de placement mais aussi en terme de mise en œuvre sur une machine.

### 3.2.2.2 La problématique traitée

Quatre classes d'applications sont définies ; par ordre de priorité [76] :

- classe 1 : les applications avec deadline.
- classe 2 : les applications à exécution immédiate (par définition prioritaires). Une date de terminaison est garantie en fonction de la prédiction calculée lors de la soumission de l'application.
- classe 3 : les applications avec ressources dédiées.
- classe 4 : les applications sans contrainte : elles s'exécutent dès que possible ou ultérieurement avec les ressources disponibles (classe "Best Effort"). Les applications irrégulières où aucune prédiction n'est possible se retrouvent ici.

Ces classes d'applications correspondent aux classes de service. Certains niveaux de qualité de service ne sont disponibles que pour certaines catégories d'applications. Dans le cas des applications pour lesquelles un modèle de prédiction du temps d'exécution existe (application régulière par exemple), la date de terminaison peut être garantie. Plus l'ordonnanceur disposera d'informations sur le support d'exécution, le réseau, les applications plus la gestion de la qualité de service sera fine.

Plusieurs files correspondant chacune à un niveau de priorité (figure 3.1) sont définies. Les files sont considérées dans l'ordre des priorités et les tâches doivent être affectées aux machines disponibles en tenant compte de ce qui est déjà placé et des contraintes.

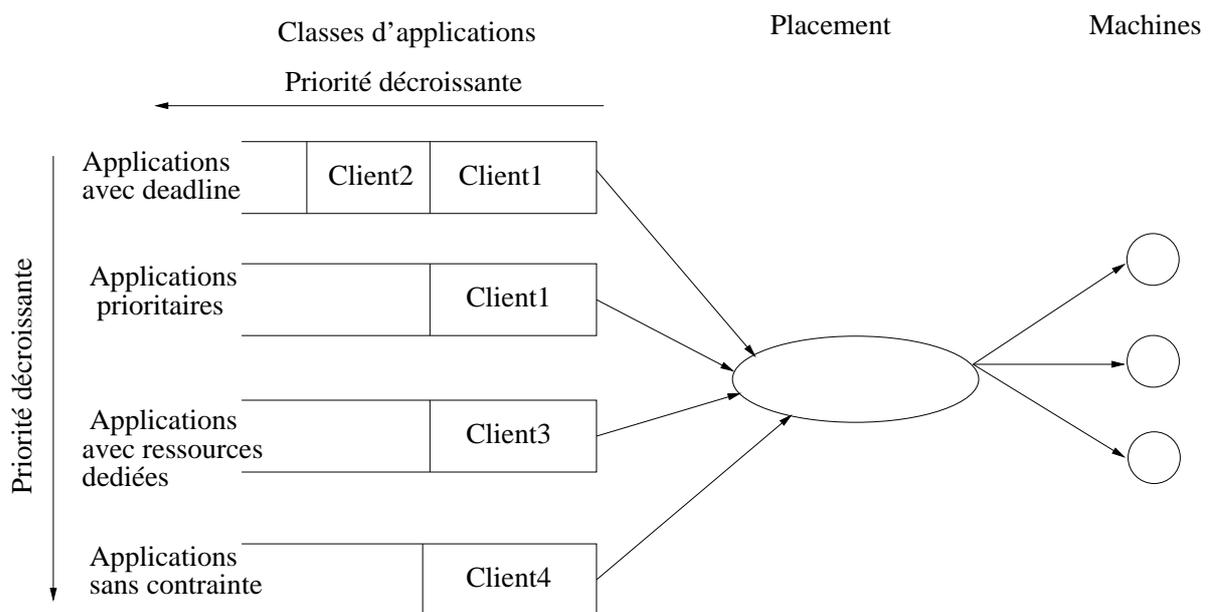


FIGURE 3.1 – Modèle de files d'attente

### 3.2.2.3 Le modèle proposé

#### *Notation*

Un événement est un début de tâche ou une terminaison de tâche.

- $M$  : nombre de machines.
- $A$  : nombre d'applications à placer.
- $N^a$  : nombre de tâches d'une application  $a$
- $X_m(t)$  : nombre de processus sur la machine  $m$  à l'instant  $t$ .
- $t_0$  : date initiale de recherche de placement.
- $t_b(a, p, m)$  : date de début d'exécution de la tâche  $p$  de l'application  $a$  sur la machine  $m$ .
- $t_f(a, p, m)$  : date de fin d'exécution de la tâche  $p$  de l'application  $a$  sur la machine  $m$ .
- $t_c(a, p, m)$  : temps processeur demandé par la tâche  $p$  de l'application  $a$  sur la machine  $m$ .
- $t_c^k(a, p, m)$  : temps de processeur encore demandé par la tâche  $p$  de l'application  $a$  après le  $k$ ème événement sur la machine  $m$ .

- $t_c(a, p, m) = t_c(a, p) * C(m)$  : équivalence de temps processeur demandé par la tâche  $p$  de l'application  $a$  en fonction de la machine  $m$ . L'utilisateur spécifie un temps  $t_c(a, p)$  calculé sur une machine d'une certaine puissance; la conversion pour déduire le temps nécessaire pour l'exécution sur la machine  $m$  est effectuée grâce au coefficient  $C(m)$ .
- $t_m^k$  : un événement d'arrivée ou de sortie de processus intervient sur la machine  $m$ . C'est le  $k$ ème événement.
- $P(a)$  : ensemble de placements possibles pour une application  $a$ .
- $t_f(m, s)$  : date à laquelle toutes les tâches prévues sur la machine  $m$  sont terminées après le placement  $s$  de l'application  $a$  avec  $s \in P(a)$ .
- $M(a, p)$  : machine sur laquelle la tâche  $p$  de l'application  $a$  est exécutée.
- $M_u(m)$  : mémoire totale utilisée sur la machine  $m$ .
- $M_t(m)$  : mémoire totale sur la machine  $m$ .
- $M_f$  : constante : coefficient de pondération.

### *Critère d'optimisation*

Plusieurs critères sont connus de la littérature : le "makespan" (minimisation de la date de fin de l'application), "sum-flow" (minimisation de la quantité de ressources utilisées), "max-stretch" (exprime le ralentissement que subit la tâche par rapport à une situation où elle serait seule sur la machine). L'objectif est d'optimiser l'utilisation des ressources et de garantir le niveau de qualité de service requis. Ainsi, il a été choisi de libérer les ressources le plus rapidement possible. Toutes les applications doivent se terminer le plus tôt possible.

$$\min_{s \in P(a)} (\max_m (t_f(m, s) + t_f(m, s) * M_u(m) * M_f / M_t(m))) \quad (3.1)$$

La date de libération des ressources est optimisée; de plus un second critère consiste à pondérer avec l'espace mémoire utilisé : les machines ayant le plus d'espace mémoire libre sont choisies en premier. En effet, pour certaines tâches l'espace mémoire demandé est inconnu donc le plus simple est de pénaliser les machines ayant le moins d'espace disponible. Le problème est multi-critères en utilisant une combinaison linéaire des différents critères. La première partie de l'addition ( $t_f(m, s)$ ) représente la date de libération des ressources de la machine. La deuxième partie de l'équation  $t_f(m, s) * M_u(m) * M_f / M_t(m)$  pénalise les machines qui ont le moins d'espace mémoire libre.  $M_u(m) / M_t(m)$  donne une idée de l'utilisation de la mémoire sur la machine. Le coefficient  $M_f$  pondère cette partie du critère. La multiplication avec  $t_f(m, s)$  positionne les valeurs dans le même ordre de grandeur que la première partie du critère.

### *Algorithme*

Un algorithme de listes est utilisé pour les applications, les tâches et les machines pour réduire la combinatoire. Le placement d'une tâche déjà étudiée est revu seulement s'il est impossible de trouver un placement pour l'application  $a$ . De plus, pour réduire le temps mis par l'algorithme, la date de libération des machines est sauvegardée et les machines sont triées par date de libération (liste\_machines) pour étudier en premier une machine qui donnerait le meilleur placement. Si le placement est possible  $t_f(m, s)$  est enregistré, et le placement sur la machine suivante est considéré; l'étude s'arrête (avec arrêt\_itération) dès qu'un  $t_m^k$  supérieur ou égal à  $t_f(m, s)$  pendant le calcul de  $t_f(a, p, m)$  est trouvé.

La qualité de service est toujours garantie : toutes les contraintes induites sont vérifiées par l'algorithme. S'il est impossible de trouver un placement la demande est refusée.

```

for application a = 1 à A applications classées par priorité do
  critère = -1
  while placement optimal non trouvé do
    if rejet=calcul de la date de début de l'application  $t_b(a, p, m)$  then
      exit
    end if
    vérification des contraintes concernant la date de début ; tâche=0
    while tâche <  $N^a$  triée par précédence ou par temps processeur demandé ET placement possible do
      numéro_machine=0 ; machine_choisie=-1
      while numéro_machine < taille(liste_machines) ET arrêt_itération == faux do
        machine=liste_machines[numéro_machine]
        vérification des contraintes concernant les machines
        arrêt_itération = prédiction de la date de fin de la tâche  $t_f(a, p, m)$  ; calcul de  $M_u(m)$ 
        if arrêt_itération == faux then
          vérification des contraintes des tâches déjà placées sur la machine
          vérification de la deadline de la tâche ; placement possible sur machine
        end if
        if (machine_choisie == -1) OU  $(t_f(m, S) + t_f(m, S) * M_u(m) * M_f / M_t(m)) < critère$  then
          machine_choisie = machine ; sauvegarde de la nouvelle valeur du critère
        end if
        numéro_machine++
      end while
      if placement possible then
        propagation des conséquences du placement
      end if
    end while
    if placement possible then
      sauvegarde de  $t_b(a, p, m)$  et  $t_f(a, p, m)$  pour toutes les tâches
      modification de l'ordre de la liste liste_machines ; placement optimal trouvé
    else
      placement optimal non trouvé
    end if
  end while
end for

```

FIGURE 3.2 – L'algorithme de placement proposé

### 3.2.2.4 Quelques résultats

Les fichiers logs de Feitelson (logs réels) ont été utilisés. Ils fournissent la date de soumission de l'application, le temps processeur demandé, le nombre de tâches. Le log `l_sdsc_sp2.swf` ([38][36], [37],) est utilisé. Le système réel a 128 noeuds. L'exécution des applications a été reproduite. L'algorithme recherche un placement sur 128 noeuds et simule leur exécution. L'influence de la qualité de service sur le temps de placement et le temps d'attente a été analysée. Quatre files sont spécifiées : low, normal, high, express et pour chaque application sa file de soumission est connue. Cette information a été utilisée pour faire la correspondance avec nos classes d'applications. Ainsi, la file "low" correspond à la classe "best effort" (classe 4), la file "normal" correspond à la classe "deadline" (classe 1), la file "high" correspond à la classe "ressources dédiées" (classe 3) et la file "express" correspond à la classe d'applications prioritaires (classe 2). Pour les applications de classe 1, leur deadline est : date de soumission + 5\*temps cpu demandé. Pour les applications de classe 2, la date de début devant être respectée est : date de soumission + 5 secondes.

Les résultats sont présentés dans le tableau 3.1. Les temps d'attente sont en secondes (s) et les temps de placement en millisecondes (ms). Les cas 1 et 4 correspondent respectivement à 10000 et 35000 événements avec les différentes classes, les cas 3 et 6 respectivement à 10000 et 35000 événements avec une classe unique "best effort".

	Cas 1	Cas 3	Cas 4	Cas 6
temps d'attente classe 1 (s)	30.512		41.98	
temps d'attente classe 2(s)	0.0036		0.0057	
temps d'attente classe 3 (s)	4845.25		5285.62	
temps d'attente classe 4 (s)	4.5564		111.14	
temps d'attente moyen (s)	710.519	64.68	396.02	70.3
temps de placement classe 1 (ms)	20.22		60.45	
temps de placement classe 2 (ms)	3.63		5.69	
temps de placement classe 3 (ms)	269.37		1463.35	
temps de placement classe 4 (ms)	7.25		233.31	
temps de placement moyen (ms)	50.39	13.08	49.19	76.39
nombre de rejets	21	0	154	0

TABLE 3.1 – Influence de la qualité de service sur les performances de l'algorithme

L'introduction de la qualité de service augmente les temps de placements et d'attente mais les performances restent bonnes (temps de placement de 50ms). Globalement l'augmentation est due aux applications de classe 3 (ressources dédiées), qui doivent attendre longtemps avant de pouvoir être seules sur les machines et l'algorithme étudie toutes les machines et une date de début pour les tâches (cela prend plusieurs itérations avant de trouver le bon  $t_b$ ). Les rejets correspondent tous à des applications deadline. Les applications de classe 2 pourraient être refusées mais un placement a toujours pu être trouvé. Les applications de classe 3 et 4 ne peuvent jamais être refusées par contre elles sont retardées.

Cet algorithme a été complété en introduisant de manière artificielle la notion de processeur virtuelle en jouant sur les priorité "nice" offertes par les noyaux UNIX. Ceci avait un sens car à l'époque les machines virtuelles commençaient tout juste et la notion de CPuset n'existait pas sous Linux.

### **Encadrement de thèse et publications :**

- Thèse de Patricia Pascal
- P . Pascal, S.Richard, B. Miegemolle, T. Monteil, Tasks mapping with quality of service for coarse grain applications, 11th International Euro-Par Conference (Euro-Par 2005), Lisbonne (Portugal), 30 Août - 2 Septembre 2005.

## **3.2.3 Ordonnancement et réseau**

### **3.2.3.1 Contexte**

Les grilles de calcul offrent l'accès à une quantité importante de ressources ce qui offre de grandes possibilités mais rend encore plus difficile le choix des machines utilisables pour une application donnée. Souvent les outils d'ordonnancement délèguent le choix du nombre de machines à réserver sur chaque site à l'utilisateur. Au mieux ils fournissent une surcouche permettant d'assurer que toutes les ressources des différents sites seront disponibles en même temps, c'est par exemple le cas avec OARGRID qui est une surcouche au dessus de OAR [23].

### **3.2.3.2 La problématique traitée**

L'objectif est de fournir un algorithme permettant de sélectionner automatiquement les machines sur plusieurs sites constituant la grille et de considérer les communications entre les tâches afin de faire les meilleurs choix. Il peut être souhaitable de répartir la charge sur plusieurs domaines tout en tenant compte des points suivants pour le placement : le même cluster doit être privilégié ; si le placement n'est pas possible, le même domaine (au sens site géographique) doit alors être un choix prioritaire ; si le placement est toujours impossible, il faut tenir compte du débit entre les sites de domaines différents.

### **3.2.3.3 Le modèle proposé**

La modélisation du réseau a un fort impact sur les temps de résolution pour obtenir un placement valide. Ceci est dû à sa complexité et les nombreuses entrées que doit avoir un modèle réseau. Afin d'avoir un placement dans un temps raisonnable, une modélisation linéaire du temps de transmission en fonction de la taille des données est choisie. Les besoins en communication d'une application sont représentés sous forme d'un graphe où les sommets représentent les tâches et les arcs la quantité d'information échangée entre deux tâches.

Dans la réalité, l'exécution d'une application se compose de plusieurs phases : calcul, communication, calcul, communication ... L'idée est de reproduire cela artificiellement afin de répartir la transmission des données durant la vie de la tâche. Ainsi, les tâches de l'application sont partagées en plusieurs sous-groupes numérotés de 1 à N (un sous-groupe correspond à une phase de calcul). Une tâche réseau virtuelle est introduite entre tout couple de tâches de calcul de l'application. C'est une tâche particulière qui ne sera pas placée ; le temps qu'elle "demande au réseau" correspond à la taille des communications. Si l'application utilise des communications de type "all to all", chaque tâche réseau sera utilisée ; sinon, certaines tâches réseau auront une taille de 0. La tâche réseau permet de représenter facilement les précédences qui existent entre une phase de calcul et une phase de communication. Une tâche de calcul de sous-groupe différent de 1 ne pourra commencer son exécution que lorsque toutes les tâches

émettrices auront terminé leur calcul et leur émission.

Sur la figure 3.3, les tâches 1', 2' et 3' constituent le premier sous-groupe. Les tâches 1'', 2'' et 3'' constituent le deuxième sous-groupe. Les temps de calcul sont partagés entre les différents sous-groupes. Par exemple, si la tâche 1 demande 500s de processeur, la tâche 1' demande 250s et la tâche 1'' demande elle aussi 250s. La tâche 1'' doit attendre que toutes les tâches qui la précèdent dans le graphe aient terminé pour commencer (c'est la même chose pour les tâches 2'' et 3'').

Le découpage de l'application en différents sous-groupes permet de synchroniser les tâches de calcul : cela évite d'avoir toute la phase de calcul puis toute la phase de communication dans le modèle (voir exemple 3.3).

Le nombre de niveaux de découpe en phase de calcul-communication est paramétrable. Toutefois si trop de niveaux sont insérés, il y a une explosion combinatoire.

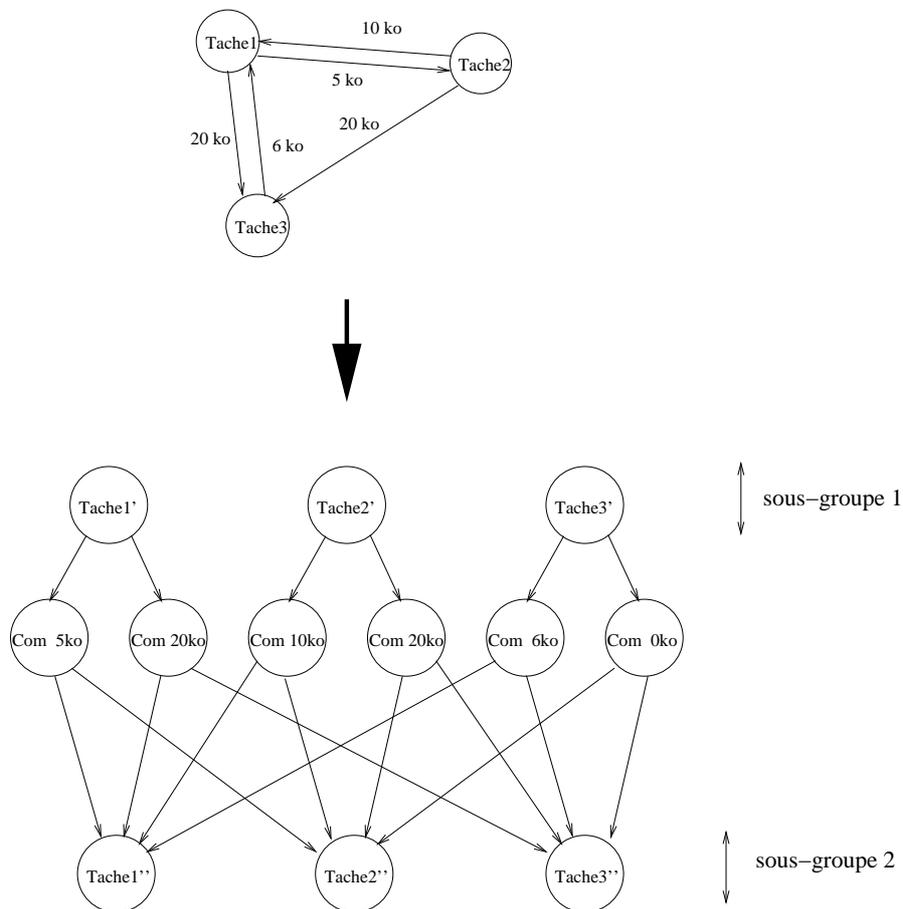


FIGURE 3.3 – Création de graphe d'application avec précedence à partir d'un graphe de tâches communicantes

L'algorithme de placement au niveau grille consiste à découper l'application reçue par l'ordonnanceur de niveau grille en groupe de tâches, à envoyer la demande sur différents domaines

puis à choisir le meilleur. Le placement au niveau domaine sera calculé par une instance locale au domaine de l'ordonnanceur qui ne connaîtra que les informations relatives à son domaine.

Pour illustrer le principe, un exemple est proposé (voir schéma 3.4). Les ordonnanceurs des domaines envoient périodiquement une information sur leur capacité de traitement prévue dans le futur en fonction des travaux qu'ils ont déjà accepté. Ceci permet à l'ordonnanceur de niveau grille de faire un classement des domaines qu'il gère.

Ensuite une boucle est exécutée tant qu'il reste des tâches à placer. A partir des données du premier domaine de la liste, un premier sous-graphe de tâches de l'application est créé. Ce groupe de tâches est placé sur plusieurs domaines de la liste (requêtes notées 1 et 3 sur l'exemple) et la date de fin d'exécution (notée 2 et 4) sur chacun des domaines est récupérée (celle-ci est calculée par une simulation du placement).

Le domaine dont la date de libération des ressources est minimale recevra un message de confirmation de placement, les autres, un message leur indiquant de supprimer le groupe de tâches.

Le nouveau groupe de tâches à placer est modifié en ôtant le sous-graphe qui vient d'être étudié.

Enfin, la liste des domaines est réorganisée en fonction du nouveau placement.

Il faut également noter que nous réessayons toujours un placement sur le domaine précédemment choisi quelle que soit sa place dans la liste ordonnée. On recherche ainsi à privilégier le placement sur un même domaine.

Dans l'exemple, le premier choix de placement concerne le domaine1 puis le domaine2.

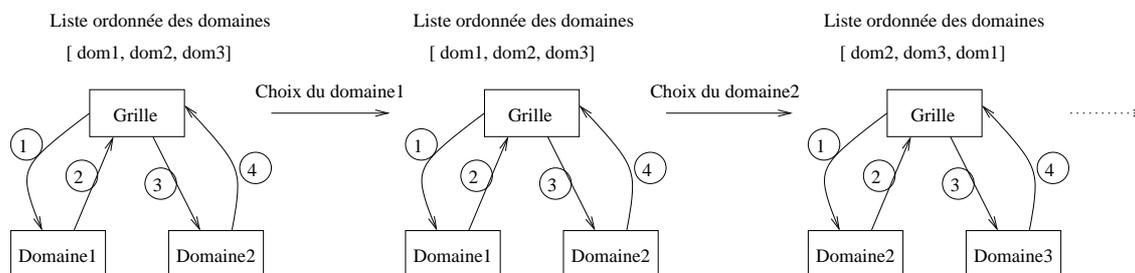


FIGURE 3.4 – Exemple simple de demande de placement au niveau grille

### 3.2.3.4 Quelques résultats

Afin de valider cette approche les ordonnanceurs de niveau grille et domaine ont été interfacés avec le simulateur de grille SimGrid [27]. L'objectif était de valider l'impact des débits offerts par le réseau. L'ensemble des tests est effectué sur une architecture au niveau grille possédant 4 domaines. Chaque domaine est composé de deux clusters de 8 machines. Deux tests sont réalisés l'un avec des débits forts entre les clusters plusieurs centaines de Mo/s et l'autre avec des débits faibles de quelques dizaines de Mo/s. 20 applications demandant en

moyenne 6 heures de calcul sont placées. Une requête de placement est faite toutes les heures. Des placements différents sont obtenus entre les débits forts et les débits faibles. 3 applications (sur les 20) ont été réparties sur plusieurs domaines pour les débits forts. Par contre pour les débits faibles chaque application a été placée sur un seul domaine. Ceci est cohérent : en effet les communications entre les domaines pénalisent la date de libération des machines. C'est en restant sur un même domaine que le placement est le plus efficace.

#### **Encadrement de thèse :**

– Thèse de Patricia Pascal

### **3.2.4 Ordonnancement et modèle économique**

#### **3.2.4.1 Contexte**

Il s'agit de proposer un modèle permettant de mettre en relation les fournisseurs des ressources de la grille avec leurs consommateurs (les utilisateurs de la grille) [40], et de réguler ainsi l'offre et la demande sur ces ressources.

Même si cela n'est pas obligatoire, il est possible que les différents fournisseurs de ressources soient également les consommateurs de ces ressources. Dans un tel cas, le modèle économique pourra être assimilé à un modèle d'échange entre les différents intervenants. La monnaie utilisée peut être, au choix, une monnaie réelle ou bien une monnaie virtuelle (jetons permettant d'acheter l'accès aux ressources). Dans tous les cas, fournisseurs et consommateurs ont chacun leurs propres objectifs qu'il convient de satisfaire au mieux. Ainsi, les premiers chercheront à maximiser leurs profits, tandis que les seconds chercheront à trouver le meilleur compromis entre la qualité de service espérée et le coût que cela représente [18].

#### **3.2.4.2 La problématique traitée**

Nous supposons que la grille est totalement dédiée aux applications placées par le biais d'un ordonnanceur basé sur le modèle économique défini ici. Il ne devra donc y avoir aucun autre moyen de lancer une quelconque application. Les grilles auxquelles le modèle économique proposé peut s'appliquer sont donc entièrement dédiées, comme cela peut être le cas dans le cadre des grilles orientées vers la prestation de services applicatifs, ou ASP pour *Application Service Provider* [8]. Cela permet d'avoir un parfait contrôle sur l'utilisation des ressources de la grille, et sur les applications qu'elles exécuteront. Enfin, s'agissant d'une grille de calcul, les ressources considérées peuvent bien sûr être hétérogènes.

Nous supposerons que toutes les tâches d'une même application sont identiques et synchrones. Elles débiteront leur exécution simultanément, et communiqueront par le biais de l'échange de messages via un réseau, et ce de manière synchrone. De plus, seules des applications à gros grain seront considérées [71] ce qui permettra de traiter grossièrement les temps réseaux.

Le modèle économique défini est basé sur un modèle de marché [19, 92, 2]. Le prix des ressources n'est pas fixe, puisqu'il peut varier au cours du temps, en fonction de l'offre et la demande par exemple. Cependant, l'évolution du prix au cours du temps est connue au moment de l'ordonnancement, afin d'inclure ces variations au résultat produit par notre modèle. Le modèle de prix inclura divers paramètres, tels que :

- le nombre de ressources utilisées,
- les caractéristiques de ces ressources,
- leur durée d'utilisation,

- le moment d’utilisation.

### 3.2.4.3 Le modèle proposé

On cherchera à minimiser la fonction objectif suivante :  $Target = W^C \cdot C_{App} + W^T \cdot T_{App}$  avec :

- $C_{App}$  = Coût de l’application. Il dépendra à la fois des processeurs choisis, ainsi que de la date de début de l’application.
- $T_{App}$  = Date de fin de l’application. Il s’agit de la date à laquelle la dernière tâche de l’application se terminera. Cette inconnue dépend également des processeurs choisis ainsi que de la date de début de l’application.
- $W^C$  et  $W^T$  sont exprimés par l’utilisateur afin de préciser ce qu’il privilégie : le coût ou la rapidité

Le temps est discrétisé en intervalles. L’estimation du temps d’une tâche sur une machine donnée est réalisée via une mini simulation événementielle prenant en compte toutes les tâches déjà placées sur cette machine c’est à dire soit en cours d’exécution soit planifiées dans le futur suite à un placement validé. Ceci permet de déterminer  $T_{App}$  [66].

Le coût de l’application est constitué de trois éléments :

$$C_{App} = C^T \cdot t^{CPU} + C^N \cdot N_U^P + C^P \cdot C_{App}^P \text{ avec :}$$

- $C^T, C^N, C^P$  sont des coefficients de pondération
- $t^{CPU}$  correspond au cumul des temps processeur réellement consommés sur la grille
- $N_U^P$  représente le nombre de processeurs utilisés par l’application
- $C_{App}^P$  donne le coût lié au type de processeurs utilisés et à la période de tarification (on suppose que le prix du processeur varie au cours du temps : par exemple moins chère le week-end ou la nuit). Son expression est détaillée dans [66].

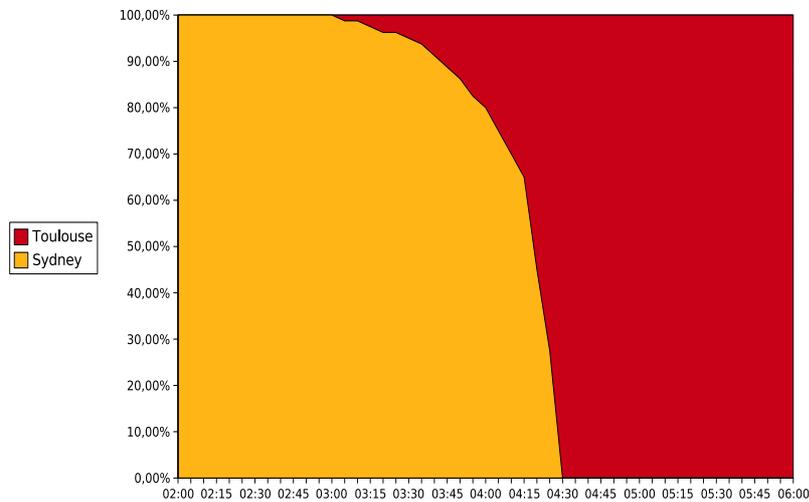
Le modèle économique présenté repose sur un problème d’optimisation non-linéaire, en variables entières, et sous contraintes. Les outils standards d’optimisation ne se prêtent que très mal à de telles caractéristiques. Une solution approchée est recherchée via un algorithme génétique [67, 48] . Les inconnues du problème sont les machines, le nombre de tâche de l’application sur la machine et la date de début. Il apparaît judicieux de représenter les individus par un vecteur dont chaque coordonnée représente la localisation de la tâche sur une machine. L’individu (2,4,1) représente l’exécution de la première tâche sur la machine 2, de la tâche deuxième tâche sur la machine 4, etc. Cette représentation facilite la mise en place des opérateurs de croisement et de mutation.

### 3.2.4.4 Quelques résultats

Dans un premier temps les résultats de l’algorithme génétique ont été comparés aux résultats donnés par le logiciel Xpress. Ceci a été possible en supprimant le choix de la date de démarrage de l’application et en la fixant à 0. Dans 90% des cas l’algorithme génétique trouve le même résultat que Xpress. Les ordonnancements trouvés dans les 10 % des cas restants ne présentent que de faibles variations par rapport à celui-ci (inférieures à 8 % de la valeur du critère). Ceci confirme le bon comportement de l’algorithme génétique.

Ensuite, l’importance du prix des ressources en fonction de l’heure a été mise en évidence. Une grille constituée de 3 clusters de 100 processeurs chacun : un à Paris (GMT+1), un à Toulouse (GMT+1) et un à Sydney (GMT+10) a été décrit. Les clusters de Toulouse et Sydney sont 2 fois plus rapides que celui de Paris en terme de processeur et sont donc plus chers à utiliser.

Les administrateurs des sites ont choisi d'augmenter le prix de leur cluster pendant les heures ouvrées (de 6 heure à 18 heure) et de diminuer le prix la nuit (18 heure à 6 heure). A cause du décalage horaire entre la France et l'Australie le cluster de Sidney devient moins cher à 9h (GMT+1) et plus cher à 21 heure (GMT+1). Un utilisateur soumet une application à 60 tâches durant chacune 12h sur un processeur à 1,2 Ghz et 6h sur un processeur à 3,8 Ghz. La date de soumission varie de 14h à 18h (GMT+1). L'utilisateur applique un poids de 10% sur le temps de restitution et 90% sur le coût financier. L'algorithme génétique propose alors des solutions qui changent en fonction de la date de soumission. Il choisit soit de démarrer tout de suite l'application ou d'attendre un peu afin de profiter de meilleur tarif. Avant 15 heure (GMT+1) Sidney est toujours préféré, de 15 heure à 16h30 (GMT+1) on bascule peu à peu sur Toulouse



(a) Proportion de choix entre Toulouse et Sydney pour des soumissions l'après midi

date soumission		15h00	15h15	15h45	16h15	16h30
Toulouse	<i>Target</i>	7800	7770	7710	7650	7620
	<i>C<sub>App</sub></i>	6720	6720	6720	6720	6720
Sydney	<i>Target</i>	7440	7560	7800	8040	8160
	<i>C<sub>App</sub></i>	6720	6840	7080	7320	7440
<i>Target</i> ≠ (%)		4.8	2.8	1.2	4.9	6.6

(b) Expérience :  $W^C = 90\%$ ,  $W^T = 10\%$

Placement avec variation de la date de soumission

FIGURE 3.5 – Simulation des placements des applications avec prise en compte d'aspect économique

### Encadrement de thèse et publications :

- Thèse de Bernard Miégemolle
- B. Miegemolle, R. Sharrock, T. Monteil, Economic Model for Grid Resource Sharing, The 2007 International Conference on Grid Computing and Applications, Las Vegas (USA), juin 2007

## 3.3 Contribution sur l'observation

### 3.3.1 Contexte

L'utilisation d'une grille sans un système d'observation est impossible. En premier lieu cela sert à l'utilisateur à avoir une vue des ressources à sa disposition et leur état plus ou moins précis. En second lieu, c'est une entrée pour tout un ensemble de services dont le placement ou bien encore le comptage de l'utilisation des ressources. Il en existe une grande variété : Network-Analyser (développé durant ma thèse) [69], NWS [93], Ganglia [65] utilisé sur grid5000, etc. Durant le projet RNTL CASP (Clusters for Application Service Provider), le logiciel Aroma a dû être développé afin de répondre à des besoins précis en terme d'intégration, de finesse des informations collectées et d'accès plus ou moins agrégé aux informations.

### 3.3.2 La problématique traitée

Le schéma d'utilisation du gestionnaire Aroma (figure 3.6) est le suivant. Des fournisseurs de services possèdent une ou plusieurs grappes de machines sur lesquelles sont installés différents noyaux de calculs d'applications demandant une importante capacité de traitement. L'utilisateur de ces applications peut, soit continuer à utiliser son application de manière traditionnelle, en exécutant les calculs sur sa machine locale soit, dans le cas de traitements coûteux en calcul, choisir de déporter le traitement sur les grappes de machines du fournisseur de services. Le client peut utiliser Aroma de deux manières différentes, soit par l'intermédiaire d'une interface graphique, soit directement à partir de son application en faisant appel à l'API d'Aroma. Dans les deux cas, la requête est transférée au fournisseur de services via le réseau local ou l'Internet. Cette transmission se fait en utilisant des mécanismes d'authentification et de chiffrement afin de préserver la confidentialité des informations émises. La requête de calcul, ainsi que les données nécessaires à la réalisation du calcul sont alors traitées par Aroma chez le fournisseur de services. Ce dernier exécute le noyau de calcul correspondant à l'application sélectionnée, sur la ou les machines choisies par un algorithme de placement. Une fois le traitement achevé, Aroma en informe le client, qui peut alors récupérer ses résultats.

Aroma s'est focalisé sur les problématiques suivantes :

- *Gestion fine des ressources* : l'ordonnancement des tâches est soumis à certaines contraintes, notamment celle de garantir un certain niveau de qualité de service. Afin d'atteindre cet objectif, l'ordonnanceur doit pouvoir s'appuyer sur une connaissance détaillée et actualisée de l'état des ressources de la grappe de machines.
- *Prise en compte de l'aspect dynamique des ressources* : une grille ou un ensemble de grappes de machines impliquent un grand nombre de nœuds. Dans de tels environnements, la défaillance ou l'ajout d'un nœud sont des événements courants qui ne doivent pas remettre en cause le fonctionnement de la structure. Pour la même raison, l'ajout de nouvelles fonctionnalités au gestionnaire doit pouvoir se faire, dans la mesure du possible, sans qu'il soit nécessaire d'arrêter le service et de re-déployer le gestionnaire sur l'ensemble des machines.
- *Facturation du service* : une trace de l'utilisation des ressources doit être conservée afin de développer des modèles économiques autour des grilles, dans le but de facturer l'utilisation du service au client.

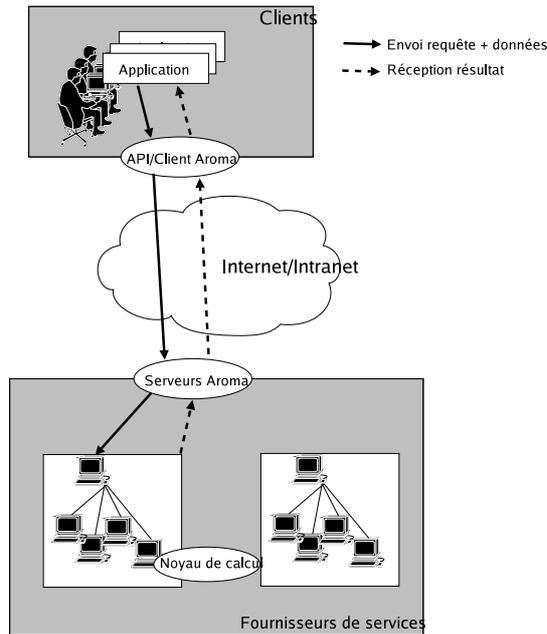


FIGURE 3.6 – Schéma d'utilisation d'Aroma

### 3.3.3 Le système proposé

#### 3.3.3.1 Système hiérarchique

Nous distinguons plusieurs niveaux hiérarchiques dans une grille, permettant d'adresser des problèmes de complexité variable :

- Le niveau grappe (Cluster) : il représente une grappe de machines, c'est à dire un ensemble de machines homogènes appartenant à un même domaine d'administration réseau. Ce niveau peut suffire à combler les besoins d'une petite équipe de personnes, par exemple un projet ou un service d'une entreprise.
- Le niveau domaine : un domaine est composé d'un ensemble de grappes appartenant au même domaine d'administration réseau. Par exemple, les différents services d'une même entreprise peuvent regrouper leurs grappes au sein d'un domaine. Le domaine leur permet ainsi de partager leurs ressources afin d'absorber des pics d'activités ou de partager des coûts de licences importants.
- Le niveau grille : ce dernier niveau permet de regrouper des domaines, et donc de mettre en commun un grand nombre de ressources n'appartenant pas au même domaine d'administration réseau. L'objectif de ce niveau est d'offrir des points d'accès vers un très grand nombre de ressources, puis de répartir les traitements vers le (ou éventuellement les) domaine(s) le(s) plus adapté(s) afin de limiter l'utilisation du réseau Internet.

La structure d'Aroma repose sur cette vision de la grille, de ce fait elle utilise une architecture hiérarchique à quatre niveaux (machine, grappe, domaine et grille). Outre le fait de tenir compte des contraintes réseau (domaines d'administration distincts), cette architecture facilite également le passage à l'échelle. En effet, les différents niveaux hiérarchiques permettent d'agréger l'information et ainsi d'éviter les effets de goulot d'étranglement.

### 3.3.3.2 Architecture logicielle

Un serveur Aroma est présent à chaque niveau hiérarchique : machine, grappe, domaine et grille. Chaque serveur Aroma est un service Jini [58] , spécialisé en fonction du niveau hiérarchique qu'il représente. Plusieurs composants sont regroupés au sein d'un serveur Aroma (figure 3.7).

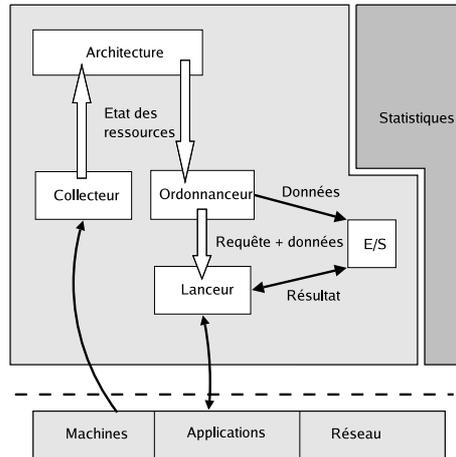


FIGURE 3.7 – Serveur générique

Le «collecteur» (*watcher*) collecte les informations sur les ressources observables (machines, réseaux) et transmet ces informations au module «architecture» qui les stocke et les utilise afin de bâtir une vue cohérente de l'architecture de la grille. L'«ordonnanceur» (*scheduler*) utilise la connaissance de l'état des ressources, fournie par le module «architecture», afin de proposer un placement. Le «lanceur» (*launcher*) exécute les décisions prises par l'ordonnanceur. Il supervise l'exécution des différentes applications (lancement, arrêt, suivi...). L'«ordonnanceur» et le «lanceur» utilisent le service «d'entrée/sortie» afin d'échanger les données nécessaires à l'exécution des applications et le résultat des calculs.

Un ou plusieurs services statistiques, gérant une base de données SQL, cohabitent avec les serveurs Aroma. Il offrent la possibilité aux serveurs Aroma d'enregistrer ou de consulter des informations représentant l'historique de l'utilisation des ressources.

Les spécificités de traitement liées à chaque niveau hiérarchique sont :

- **Niveau machine :** Un serveur de niveau machine appelé *Host Resource Unit* (HRU) est présent sur chaque machine de la grille. Il collecte toutes les informations disponibles au niveau d'une machine : charge de la machine (processeur, mémoire, etc), charge du réseau vue par la machine, les applications et processus s'exécutant sur la machine et les utilisateurs de la machine. Il n'y a pas d'ordonnanceur à ce niveau, le «lanceur» exécute les ordres provenant des ordonnanceurs de niveau supérieur.
- **Niveau grappe de machines :** Une machine parmi celles constituant la grappe héberge le service de niveau grappe appelé *Cluster Resource Unit* (CRU). Le CRU possède une connaissance agrégée de l'état de la grappe. Des décisions de placement peuvent être prises à ce niveau.
- **Niveau domaine :** Comme pour le niveau grappe, une machine appartenant au domaine héberge le service de niveau domaine appelé *Domain Resource Unit* (DRU). Le domaine est a priori géré par un administrateur unique. On retrouve les mêmes fonc-

tionnalités que sur la grappe, simplement les informations recueillies seront encore plus synthétiques.

- **Niveau grille** : Les entités de niveau grille appelées *Grid Resource Unit* (GRU) représentent des points d'accès à un ensemble de domaines et sont localisées sur ces différents domaines. Le GRU possède une vue très agrégée de l'état de la grille. L'ordonnanceur d'un GRU peut, soit prendre une décision de placement, soit transférer la requête vers un de ses domaines qui se chargera de placer l'application sur ses machines.

Une même machine physique peut cumuler plusieurs fonctions : HRU+CRU par exemple. Dans ce cas les deux serveurs Aroma, utilisés partagent la même machine virtuelle Java afin de ne pas surcharger la mémoire de la machine.

### 3.3.3.3 Le module collecteur

La difficulté est de collecter cette information sans surcharger exagérément les machines. Plusieurs types d'information sont nécessaires :

- **Données statiques** : ces données sont dites statiques car elles n'évoluent pas durant la durée de vie du gestionnaire. Elles concernent la version du système d'exploitation, le nom de la machine, la date de dernier démarrage... Ces données sont collectées une fois pour toute lors de la création du service Aroma.
- **Données dynamiques** : ces données donnent une image de la variation de l'état des ressources au cours du temps. Elles nécessitent un rafraîchissement périodique et automatique. La durée de la période de rafraîchissement des données a une grande importance. En effet, ces données sont directement utilisées lors de la prise d'une décision de placement de tâches. De l'exactitude de ces mesures dépendra la qualité de la décision de placement prise.
- **Données structurelles** : ces données concernent les informations sur la structure de la grille à un instant donné. Cette architecture peut évoluer notamment à cause des pannes. Il est nécessaire pour les différentes entités d'avoir des informations sur l'architecture afin de savoir avec quelle nouvelle entité elles doivent communiquer.
- **Données statistiques** : les données collectées peuvent servir à constituer un historique de l'utilisation des ressources de la grille. De nombreuses moyennes sont réalisées avec des fenêtres temporelles différentes (une heure, un jour, une semaine, un mois, six mois, un an, etc). Ces données sont conservées dans une base de données et ont une durée de vie proportionnelle à la taille de leur fenêtre temporelle.

### 3.3.3.4 Chargement dynamique de capteur de charge

Pouvoir modifier la liste des informations collectées sur les machines de la grille, ou modifier la manière dont certaines informations sont collectées, sans être obligé de stopper le fonctionnement de la grille est une fonctionnalité intéressante. Nous utilisons ici la possibilité de chargement dynamique de classes offerte par le langage Java. Chaque serveur de la grille utilise des fichiers de configuration dans lesquels sont décrites les ressources à observer.

Le module «collecteur» s'appuie sur ces fichiers afin de construire et instancier chaque capteur de charge requis. La lecture des fichiers de configuration est effectuée d'une part au démarrage du serveur pour construire les capteurs initiaux, puis en cours d'exécution pour charger dynamiquement les nouveaux capteurs via une interface d'administration.

### **Encadrement de thèse et publications :**

- Thèse de Samuel Richard
- P. Pascal, S. Richard, T. Monteil, “Architecture of a grid resource manager”, International conference on Parallel and Distributed Processing Techniques and Applications, Las-Vegas, june 24-27, 2002

## **3.4 Bilan**

Pour solutionner les problèmes étudiés, nous avons utilisé des algorithmes d’exploration de systèmes combinatoires, l’optimisation non linéaire en variables entières sous contraintes, les algorithmes génétiques, la simulation de grille, l’architecture logicielle en JAVA et le dialogue avec les systèmes d’exploitation. Ceci a permis de proposer de nouveaux algorithmes de placement qui sont tout à fait d’actualité par exemple dans le cadre du "cloud computing" qui a besoin éventuellement de partager ses machines avec dessous des modèles économiques qui pourraient être plus évolués que ce qui est proposé à l’heure actuelle.

L’architecture proposée dans Aroma lui permet de répondre à la problématique de l’observation des grilles. Ses capacités de chargement à chaud de capteur de charge lui permettent aussi d’assurer un service de qualité et permettent d’alimenter les algorithmes de placement. Ces travaux devraient avoir des prolongements dans des projets liés à la gestion de centre de données et de calcul sous des contraintes de QoS, d’énergie et de rentabilité.



## Chapitre 4

# La couche intergiciel

### 4.1 Introduction

La couche intergiciel fait le lien entre l'utilisateur et les ressources informatiques. Elle s'appuie sur la couche service. Les intergiciels permettent d'offrir à l'utilisateur des fonctionnalités complexes utilisables relativement facilement. Elle comble les manques des systèmes d'exploitation et offre une souplesse d'innovation que l'on ne peut pas avoir dans les systèmes d'exploitation.

Durant mes activités de recherche, j'ai été amené à créer ou participer à la création de trois intergiciels conséquents. Le premier était un environnement de programmation parallèle (LANDA) avec une participation à ces deux dernières versions, le second était un environnement permettant de mettre en place et de gérer la mise en mode ASP (Application Service Provider) d'applications sur des grilles et cluster (Aroma) enfin une partie de ces dernières années ont été consacrées à l'enrichissement d'un environnement de déploiement et de gestion autonome (TUNe).

### 4.2 Contribution sur les environnements de programmation parallèle

#### 4.2.1 Introduction

LANDA, Local Area Network for Distributed Applications, est à la fois une bibliothèque de communication par échange de messages et un environnement graphique intégré. Comme PVM, il permet l'exécution d'une application distribuée sur un ensemble de machines. Ce projet a débuté au LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes) en 1989. Il a été réalisé grâce aux soutiens tant financiers que matériels qu'ont apportés le conseil régional Midi-Pyrénées et les sociétés SUN et IBM.

#### 4.2.2 Présentation des différents modules de LANDA

LANDA est constitué d'une série de modules [70] coopérant les uns avec les autres. Les plus importants sont :

- Le noyau construit sur RPC-XDR pour le contrôle de l'application et la gestion des messages.

- L'environnement graphique intégré pour l'édition et le monitoring des applications.
- La bibliothèque de communication de haut niveau.
- Le module d'analyse et de mesure de performance d'une application.
- Le module de surveillance de la charge de la machine virtuelle LANDA.

### 4.2.3 Les fichiers

LANDA a besoin d'informations sur la machine virtuelle et sur l'application de l'utilisateur. Pour cela, il utilise des fichiers de données. Il y a le fichier *network file* qui décrit la structure physique de la machine parallèle virtuelle (technologie, hiérarchie,...). Il est écrit par le ou les administrateurs réseau.

L'utilisateur va créer manuellement ou graphiquement le fichier *configuration file*. Il caractérise son application. Il spécifie entre autres les tâches initiales et leurs machines, les groupes de communication au lancement, les règles de terminaison d'une application. Les deux derniers fichiers : *schedule file* et *statistic file* sont construits lors du déroulement d'une application. Ils sont optionnels et représentent respectivement une trace des événements lors du déroulement et les performances de l'application parallèle.

### 4.2.4 Le noyau

Le système de communication est réalisé à l'aide de trois familles de serveurs :

- Le serveur d'application gère l'application. Ceci est assuré par la gestion des événements (envoi, réception, création, terminaison, ...). Il exécute l'ensemble de l'application et détecte les erreurs ainsi que la fin de l'application.
- Les serveurs pour la communication, propre à chaque utilisateur, réalise le routage des messages. Il s'agit ici de gérer les boîtes aux lettres et de distribuer les messages. Ceci est réalisé pour chaque application par un serveur d'entête et des serveurs de données.
- Les serveurs de tâches uniques sur chaque machine contrôlent les tâches de tous les utilisateurs de la machine. Ces serveurs ont en charge le lancement des processus. Ils réveillent les tâches en attente de lecture bloquante, transmettent les mises à jour des données globales et détectent la terminaison des tâches.

### 4.2.5 Les interfaces graphiques

LANDA peut fonctionner en mode batch pour des besoins de performance, ou bien en mode graphique pour le débogage et l'analyse de son application. Cet environnement est constitué d'outils indépendants qui offrent de multiples fonctionnalités pour créer, modifier, superviser, analyser des applications parallèles et visualiser la charge de la machine virtuelle. En voici les principales composantes :

- LANDA :

La fenêtre principale de LANDA (figure 4.1) est une interface ergonomique pour superviser une application. Le contrôle est un des aspects importants grâce à une vision instantanée de l'état des tâches de l'application. L'environnement est composé de 4 parties :

- Un ensemble de boutons de commandes pour appeler les outils d'analyse de LANDA (History, Statistic, Net Analyse) et quitter LANDA.
- Une partie pour charger, éditer, sauvegarder et vérifier une application parallèle.

- Une partie pour lancer, arrêter et paramétrer une application (événement, animation, ...).
- Une fenêtre graphique pour visualiser et éditer l'application. L'animation permet de suivre le déroulement de l'application à travers les événements (envoi et réception de messages, création de tâche, fin de tâche, ...) et de connaître la liste des messages en attente de lecture dans chaque boîte aux lettres.
- History :  
Cet utilitaire trace les événements d'une application en temps réel ou en post-traitement et en affiche les caractéristiques (date, durée, type d'événement, paramètres, ...). Cet outil est intéressant pour analyser le comportement d'une application parallèle. La liste des événements peut être sauvée dans un fichier.
- Statistic :  
Cette interface graphique présente sous différentes formes les données statistiques déduites des événements d'une application (efficacité, accélération, temps global, ...).
- Net Analyse :  
Cet outil visualise les courbes de charge des processeurs (stations de travail) de la machine virtuelle avec une mémorisation de 24 heures de mesure. Les données sont obtenues par les serveurs de charge installés sur les stations de travail de toute la machine virtuelle. Ce système est optionnel. Ces informations sont primordiales pour aider l'utilisateur à choisir les processeurs sur lesquels il veut exécuter son application ; elles sont aussi intéressantes pour les administrateurs système pour connaître le comportement général du réseau.

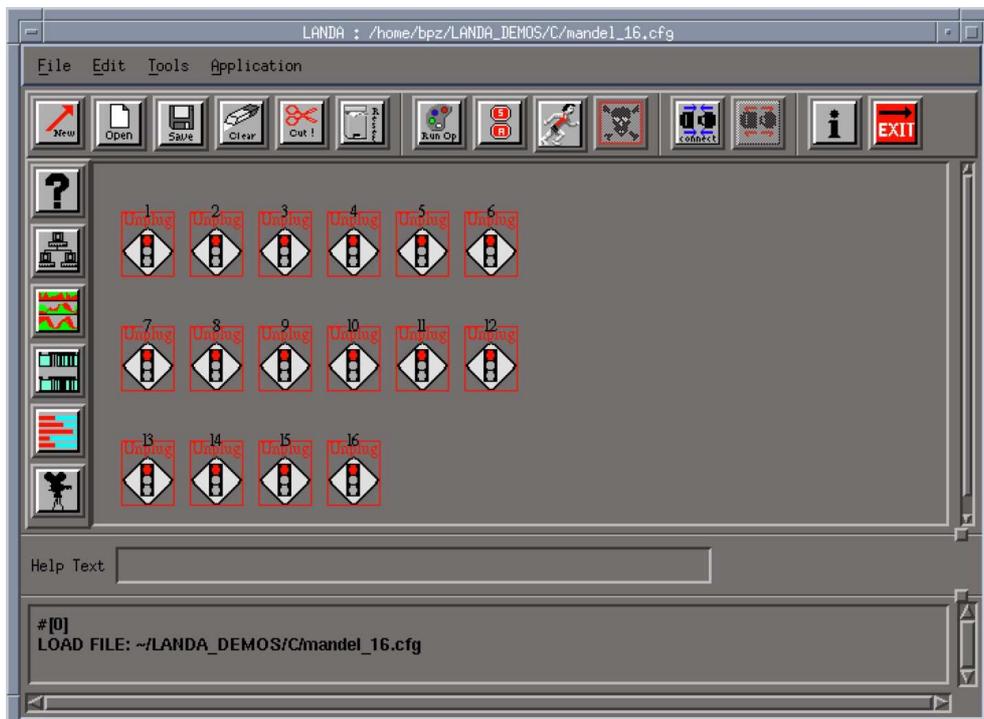


FIGURE 4.1 – Fenêtre principale de LANDA

## 4.2.6 Utilisation de la bibliothèque

La bibliothèque de communication de LANDA est écrite en C et est interfacée avec les langages Fortran et Pascal. Elle permet de façon transparente à l'utilisateur la communication synchrone ou asynchrone par échange de messages. Elle est compatible avec MPI et PVM.

La bibliothèque fournit un ensemble de fonctions pour créer dynamiquement des tâches et connaître les paramètres et l'état de l'application (nombre de tâches, programme, ...).

## 4.2.7 Ma contribution

Ma contribution à ce projet a été de concevoir et implémenter le système de communication de la version 2 (bâtie sur un système centralisé de communication) et de la version 3 (bâtie sur une couche d'abstraction des supports de communication et sur un système distribué de stockage des données), de fournir un outil d'observation de charge, de créer des algorithmes de placement et l'outil associé et enfin de diriger l'ensemble de l'équipe de développement sur la version 3.

### Encadrements de thèse et publications :

- Thèse de David Gauchard
- Thèse de Thierry Monteil
- D. Gauchard, T. Monteil, J.M Garcia, O. Brun, C. Fournié, B. Lecussan "Communication mechanisms for Myrinet and MPC in the Parallel Environment LANDA-HSN". First Myrinet User Group Conference MUG2000, Lyon, septembre 2000.
- T. Monteil, J.M. Garcia, P. Guyaux, "LANDA : une machine virtuelle parallèle", revue calculateurs parallèles, Hermès, volume 7, No 2, 1995.

## 4.3 Contributions sur les environnements pour les fournisseurs de service

### 4.3.1 Introduction

L'utilisation du gestionnaire de ressources dans un mode ASP (*Application Service Provider*) soulève de nouveaux problèmes. Notre travail se positionne dans un contexte mettant en jeu deux acteurs principaux : un client désirant exécuter un calcul complexe et un fournisseur de services mettant à disposition ses machines et ses applications. Dans ce contexte d'utilisation, un contrat passé entre chaque client et le fournisseur de services, doit permettre de définir les attentes du client et les modalités de facturation du service vis à vis de ce client. La confidentialité des données échangées, des codes de calcul et l'authentification des clients sont également des problèmes à prendre en considération. De même, la facilité de portage d'applications existantes, en mode d'utilisation ASP, est un aspect à ne pas négliger afin de ne pas rebuter les utilisateurs potentiels.

### 4.3.2 La notion de contrat

Le contrat, fruit de négociations entre le client et le fournisseur de services, permet de définir le cadre d'utilisation des services. Il permet de définir trois types d'informations :

- des limites concernant l'utilisation que peut faire le client des services,
- le niveau de qualité de service garantie au client,

– le mode de facturation du service.

Définir un contrat particulier pour chaque utilisateur de la grille serait une opération fastidieuse et inutile. Afin d'éviter cela, Aroma permet d'associer un ensemble d'utilisateurs à un même contrat. Chaque groupe d'utilisateur est ensuite libre de répartir les ressources dont il dispose entre les différents membres du groupe. Pour ce faire, tout contrat possède un ou plusieurs utilisateurs particuliers chargés d'administrer le contrat. L'ensemble des informations associées au contrat sont stockées dans une base de données centralisée, qui est traitée par le *DataManager service*.

### 4.3.3 La sécurité

La sécurité concerne à la fois l'authentification des utilisateurs et des fournisseurs de services, le chiffrement des communications, la non-répudiation et la sécurité des données et des codes de calcul.

#### 4.3.3.1 Authentification des utilisateurs.

Aroma utilise une authentification des utilisateurs par certificat numérique *X509*. Chaque utilisateur du gestionnaire doit posséder un certificat numérique qui permet de l'authentifier. Lors d'une connexion au gestionnaire de ressources, le mot de passe du certificat doit être saisi afin de poursuivre tout dialogue avec les services. L'authentification repose sur l'utilisation de JAAS<sup>1</sup> (Java Authorization and Authentication Service), ce qui permet de modifier facilement le type d'authentification utilisé.

#### 4.3.3.2 Chiffrement des communications

La version de Jini utilisée (v1.2) pour l'accès aux services d'Aroma n'offre aucun mécanisme de sécurité. De ce fait, une solution partielle basée sur l'utilisation de JSSE<sup>2</sup> (Java Secure Socket Extension) et du protocole TLS (Transport Layer Service) a été mise en place. Cette solution assure l'authentification mutuelle entre le client et le serveur, ainsi que le chiffrement des communications réalisé grâce aux clés publiques et privées de chaque protagoniste.

#### 4.3.3.3 Protection des données et du code.

Les données et les codes de calcul doivent être également protégés de malveillances pouvant provenir d'utilisateurs locaux à chaque machine d'exécution. Ces malveillances peuvent provenir d'un utilisateur se connectant directement sur la machine, sans passer par le gestionnaire de ressources, ou d'un client du fournisseur de services utilisant un code de calcul espion cherchant à pirater des informations appartenant à d'autres clients du fournisseur de services, ou au fournisseur de services lui-même.

Chaque client ASP est associé à un utilisateur Unix sur les machines d'exécution. Les données sont stockées dans un répertoire appartenant à l'utilisateur Unix en question, et lisibles uniquement par cet utilisateur. De même, lors de son exécution, le code de calcul appartient à ce même utilisateur Unix. Ces mécanismes permettent d'assurer le même niveau de protection que sur une machine standard.

---

1. <http://www.java.sun.com/JAAS>

2. <http://www.java.sun.com/JSSE>

#### 4.3.4 Interactions entre un client et le gestionnaire de services

Deux modes d'utilisation d'Arora sont proposés. Un premier mode permet d'invoquer le gestionnaire à partir d'une API (Application Programming Interface), et ainsi d'intégrer le mode ASP dans un logiciel pré-existant. Le deuxième mode, quant à lui, est utilisable via une interface graphique qui offre l'accès à l'ensemble des fonctionnalités du gestionnaire de ressources.

##### 4.3.4.1 Le client Arora : problématique et avantages

La partie cliente du gestionnaire de ressources doit permettre de faire appel à l'ensemble des fonctionnalités offertes par le gestionnaire, durant toute la durée de vie de ce dernier. De plus, elle ne doit pas nécessiter l'utilisation d'un nombre trop important de ressources chez le client, afin de pouvoir être utilisée à partir de simples terminaux ou même par l'intermédiaire d'un smartphone, par exemple.

Les fonctionnalités du gestionnaire de ressources pouvant être étendues au cours du temps, le client Arora repose sur l'utilisation d'une API générique, qui permet d'invoquer un traitement à partir d'un numéro et d'une liste de paramètres de taille et de types variables. De cette manière, un client ayant intégré le mode ASP à son logiciel de calcul, pourra s'il le souhaite, modifier son logiciel afin de bénéficier des nouvelles fonctionnalités offertes, sans avoir besoin d'acquiescer un nouveau logiciel.

Dans le même but, l'interface graphique d'Arora offre la possibilité de télécharger automatiquement, via le réseau, de nouveaux *plug-ins* graphiques permettant ainsi d'accéder à de nouvelles fonctionnalités ou d'améliorer les fonctionnalités existantes.

##### 4.3.4.2 Définition des *plug-ins* graphiques

Un *plugin* graphique correspond à un module disponible dans l'interface graphique cliente d'Arora. Il contient l'interface graphique de ce service, mais également les moyens de dialoguer avec les serveurs en vue de réaliser le service.

Dans Arora, il existe sept *plug-ins* différents :

- *Services Launcher* : ce service correspond au conteneur de services de l'interface cliente. Il est nécessaire au lancement de cette dernière.
- *Application Launcher* : il permet à l'utilisateur de lancer ses propres applications sur les machines gérées par Arora. Le nom de l'exécutable, les paramètres de l'application et la qualité de service à garantir peuvent être choisis via l'interface graphique.
- *Resources Observer* : ce service permet de visualiser l'état des ressources disponibles sur chaque machine à laquelle le client est connecté.
- *Configuration File Editor* : il permet à l'administrateur de configurer l'architecture de la grille gérée par Arora, c'est-à-dire de choisir sur quelles machines lancer des serveurs Arora et de fixer le type de ces serveurs (grille, domaine, grappe ou machine).
- *Users Rights Editor* : il est utilisé par l'administrateur pour gérer les droits accordés à chaque utilisateur en terme d'utilisation de services ou bien de consommation de ressources.
- *Statistic* : ce module offre à l'utilisateur un historique de l'utilisation de la grille en termes de ressources consommées ou bien d'applications exécutées.
- *Administration Service* : il permet à un administrateur de la grille de démarrer, d'arrêter ou de visualiser l'état courant d'une grille ou d'un sous-ensemble de machines d'une grille.

Chacun de ces services est accessible à partir de l'interface graphique cliente, sauf le *Services Launcher* qui correspond à l'interface graphique elle-même.

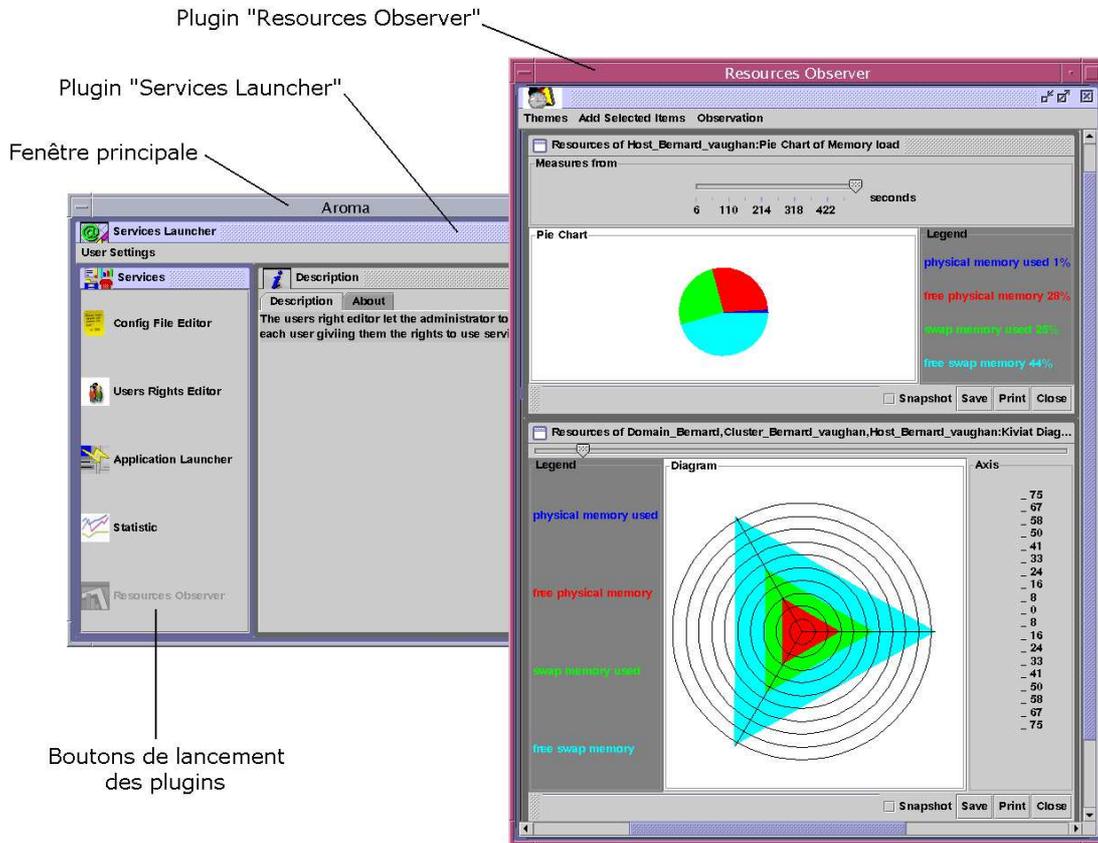


FIGURE 4.2 – Interface graphique cliente

#### 4.3.4.3 *Plugins* et droits utilisateurs

Lors d'une connexion à une grappe par l'intermédiaire de l'interface graphique, la phase d'authentification est suivie par une phase de récupération de la liste des *plugins* graphiques utilisables par l'utilisateur authentifié. Chaque *plugin* graphique est stocké dans une archive JAR, que le client doit posséder sur sa machine afin d'instancier la classe Java correspondante.

Le principe de fonctionnement est très simple. Une table de la base de données d'Aroma contient, pour chaque utilisateur, la liste des *plugins* graphiques qu'il est autorisé à utiliser. Lorsque l'interface se connecte à un serveur, celui-ci accède à la base de données et récupère la liste des *plugins* graphiques pour l'utilisateur connecté. Cette liste est transmise au client qui vérifie s'il possède bien toutes les archives JAR nécessaires et si les versions des *plugins* qu'il possède ne sont pas obsolètes. Dans le cas où un *plugin* graphique est absent ou trop ancien, ce dernier est téléchargé depuis le serveur sur le poste client. Les technologies ont évolué depuis et il existe des méthodes répandues pour réaliser cela basées sur les travaux de l'OSGI (Open Services Gateway initiative) par exemple.

### 4.3.5 Ma contribution

Aroma faisait partie du projet RNTL CASP dont j'étais le coordinateur. Ce logiciel a été la réponse proposée pour satisfaire les besoins de CASP pour gérer l'ASP. Il a été conçu et développé au LAAS lors de ce projet sous ma direction avec l'aide de deux doctorants que j'encadrais et des stagiaires de niveau Master.

#### Encadrement de thèse et publications :

- Thèse de Samuel Richard
- P. Bacquet, O. Brun, J.M. Garcia, T. Monteil, P. Pascal, S. Richard, Telecommunication Network Modeling and Planning Tool on ASP clusters, International Conference on Computational Science 2003 (ICCS'2003), Saint Petersburg, Russie et Melbourne, Australie, 2 au 4 Juin 2003, pp 514-523.

## 4.4 Contribution sur les environnements autonomiques

### 4.4.1 Contexte

Le but de l'informatique autonome est de surpasser la complexité des systèmes logiciels et matériels utilisés dans l'informatique d'aujourd'hui et de demain en leur procurant des capacités d'auto-gestion. Les environnements autonomiques sont une expression de ce concept.

Pour atteindre les objectifs d'une informatique autonome, l'idée principale est de s'inspirer des boucles de contrôles utilisées en automatique : un contrôleur guidé par un modèle ajuste en continu des paramètres sur les objets contrôlés en fonction des mesures de retour. IBM a suggéré un modèle de référence pour la boucle de contrôle autonome en informatique [60], qui est souvent appelé la boucle MAPE-K (**M**onitor, **A**nalyse, **P**lan, **E**xecute, **K**nowledge). Dans la boucle MAPE-K, les **éléments gérés** représentent des ressources logicielles ou matérielles auquel un comportement autonome est fourni en les couplant avec un **gestionnaire autonome**. Les **capteurs**, aussi appelés senseurs, sondes ou probes, collectent des informations, aussi appelées métriques, sur les éléments gérés. Les actionneurs prennent en charge les changements à effectuer sur les éléments gérés. Nos travaux se sont déroulés dans le cadre du projet TUNe (Toulouse University Network) [13] développé à l'IRIT sous la direction de Daniel Hagimont. Cet environnement fournit un cadre non intrusif pour déployer, démarrer et gérer des logiciels distribués. Nos travaux se sont concentrés sur son adaptation à une grille et l'ajout de fonctionnalités pour gérer finement la performance des logiciels administrés de manière autonome.

### 4.4.2 Le logiciel TUNe

#### 4.4.2.1 Principe

La gestion des logiciels est basée sur la boucle de contrôle autonome [10]. Chaque aspect de l'administration autonome (déploiement, configuration, démarrage, reconfiguration) a des besoins spécifiques en terme de description. Quatre types de langage spécifique (Domain Specific Modeling Language - DSML) existent : un pour décrire l'architecture matériel (Hardware Description - HD), un pour expliciter l'architecture logiciel (Software Description - SD), un pour réaliser la connexion non intrusive entre TUNe et le logiciel à gérer (Software Wrapper Description - SWD) et un pour décrire les politiques de gestion (Policy Description - PD).



FIGURE 4.3 – HD pour la description matérielle pour Grid'5000

C'est ce dernier type de représentation (PD) qui est le résultat de nos travaux. Les diagrammes (HD, SD et PD) peuvent être créés graphiquement avec des outils comme Topcased [35] car ils sont basés sur la syntaxe UML. La particularité de TUNe est de faire des descriptions en intention. Le contexte au moment du lancement de l'application (état du matériel par exemple) crée une instantiation spécifique des différents diagrammes. L'utilisation de TUNe se résume à :

- création des différents diagrammes : HD, SD, PD et du SWD
- lancement de TUNe avec un lien sur les différentes descriptions
- déploiement initial qui consiste à instancier la description de l'architecture logiciel sur l'architecture matériel
- création des différents fichiers de configuration propres au logiciel à administrer
- transfert de tous les données sur les différentes machines choisies
- démarrage de l'application
- gestion autonome par TUNe
- arrêt, nettoyage et rapatriement des résultats

#### 4.4.2.2 L'architecture matériel

Ces diagrammes sont basés sur les diagrammes de classes en UML. Chaque classe correspond à une famille de matériel. La figure 4.3 représente deux familles de machines pour la grille GRID'5000. Des informations comme le "login" (*rsharroc*), le répertoire de déploiement sur chaque machine (*DirLocal*) créé automatiquement par TUNe, la localisation du jdk java (*javahome*) ou bien encore des arguments spécifiques pour le gestionnaires de ressources (*type*) sont fournies.

#### 4.4.2.3 L'architecture logiciel

L'intergiciel DIET [25] est utilisé comme exemple pour décrire les SD. Ces derniers sont bâtis sur les diagrammes de classes UML. DIET est un outil distribué pour accéder à des services de calcul. Il est bâti sur une architecture hiérarchique. Dans le SD (figure 4.4), chaque classe représente un type d'exécutable à instancier. Une cardinalité maximum et minimum entre les différentes classes permet de limiter le nombre d'instances. Dans le SD représentant l'architecture de DIET, apparaît différents types d'agent : les Master Agents (MA) qui sont

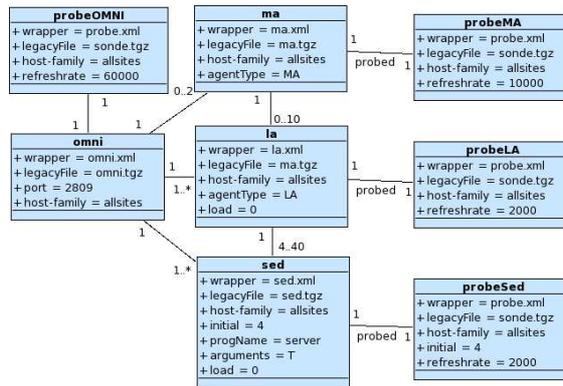


FIGURE 4.4 – SD pour une architecture DIET

connectés aux Local Agents (LA) qui gère des SErver Daemons (SED) chargés d’offrir les services de calcul. Les communications entre les agents sont administrées via un système CORBA : omniORB system (OMNI). Des sondes (ou probes) sont accrochées à l’OMNI, aux MA, aux LA et aux SED. Elles observent la charge et l’état des processus. Des attributs pré-définis sont fournis par TUNE : le nombre initial d’instances (*initial*), la famille des machines (*host-family* qui correspond au nom dans le HD), l’archive contenant le logiciel (*legacyFile*).

#### 4.4.2.4 La connexion entre TUNE et l’application

Les différents parties de l’application sont représentées par des composants Fractal [14]. La répercussion des actions de la représentation logique vers le système réel se fait à partir d’un "wrapper". Chaque action est décrite dans un fichier XML. Il contient des méthodes pour fabriquer les fichiers de configuration, pour démarrer ou arrêter une application, etc. Voici un extrait du SWD des SED de DIET :

```

<?xml version='1.0' encoding='ISO-8859-1' ?>

<wrapper name='sed'>

  <method name="start" key="extension.GenericUNIXMethods"
method="start_with_pid_linux" >
    <param value="$dirLocal/$progName $dirLocal/$srname-cfg $arguments $srname"/>
    <param value="LD_LIBRARY_PATH=$dirLocal"/>
    <param value="OMNIORB_CONFIG=$dirLocal/$omni.srname-cfg" />
  </method>

  <method name="configureOmni" key="extension.GenericUNIXMethods"
    method="configure_plain_text">
    <param value="$dirLocal/$omni.srname-cfg"/>
    <param value=" = "/>
    <param value="InitRef:NameService=corbaname::$omni.nodeName:$omni.port"/>
    <param value="DefaultInitRef:corbaloc::$omni.nodeName" />
  </method>

```

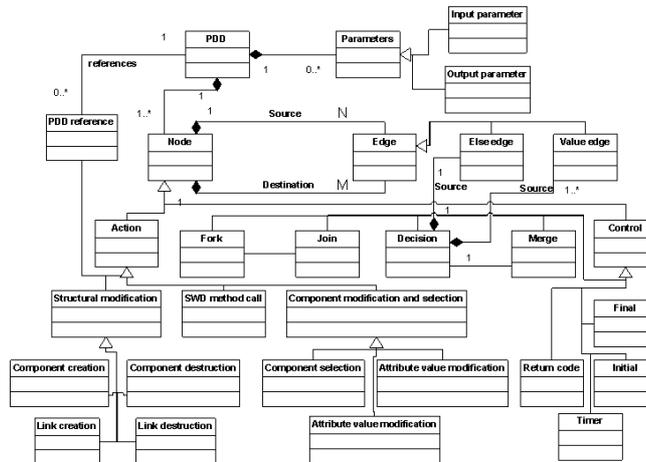


FIGURE 4.5 – Méta-modèle des diagrammes PD

...

### 4.4.3 Les politiques de gestion (PD)

Les diagrammes de gestion des politiques ont été modélisés par un méta-modèle basé sur les diagrammes d'activités UML (figure 4.5). Ce dernier est composé de deux grands stéréotypes de noeuds (les actions et les contrôles) pouvant avoir des entrées/sorties. Les arcs représentent des transitions possibles d'un noeud vers un autre. Les actions décrivent une manipulation sur les systèmes gérés et les noeuds de contrôle permettent des choix sur le chemin d'exécution. Ces deux stéréotypes sont ensuite dérivés afin de les spécialiser [82]. Concernant les actions, ils contiennent une expression dont la syntaxe suit une formulation étendue de Backus-Naur (EBNF) qui permet de mettre en place un ensemble d'opérateurs (un extrait est donné dans la figure 4.6). Par exemple, un composant issu du SD est créé avec l'opérateur ++ avec en

```

component creation = list , "=" , SD class name , "++" ,
  [ "[" , number of instances to create , "]" ] ;
PD reference = [ outs ] , "=" , PD name , "(" , [ ins ] , ")" ;
outs = out , { "," , out } ;
out = list , { " " , variable } ;
ins = in , { "," , in } ;
in = list , { " " , input } ;
input = variable | attribute ;

```

FIGURE 4.6 – Exemple d'expression des opérateurs

option le nombre d'instances à créer (par défaut une). Un autre exemple permet à partir d'un PD l'appel d'un autre PD avec d'éventuels paramètres en entrée/sortie.

Les PD peuvent ensuite être utilisés pour démarrer l'application, réaliser des actions de création ou bien encore gérer les performances de l'application (figure 4.7). Dans ce dernier, un nouveau SED est automatiquement créé et accroché à un LA chargé à moins de 80% s'il en

existe un sinon TUNe le créé automatiquement.

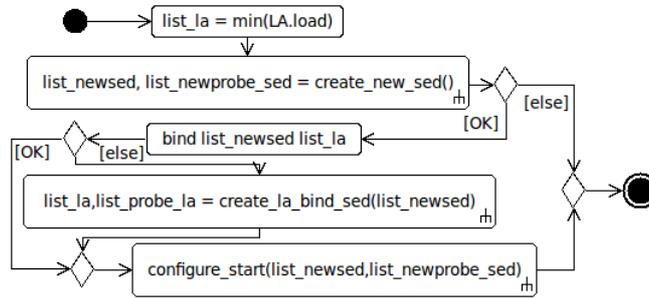
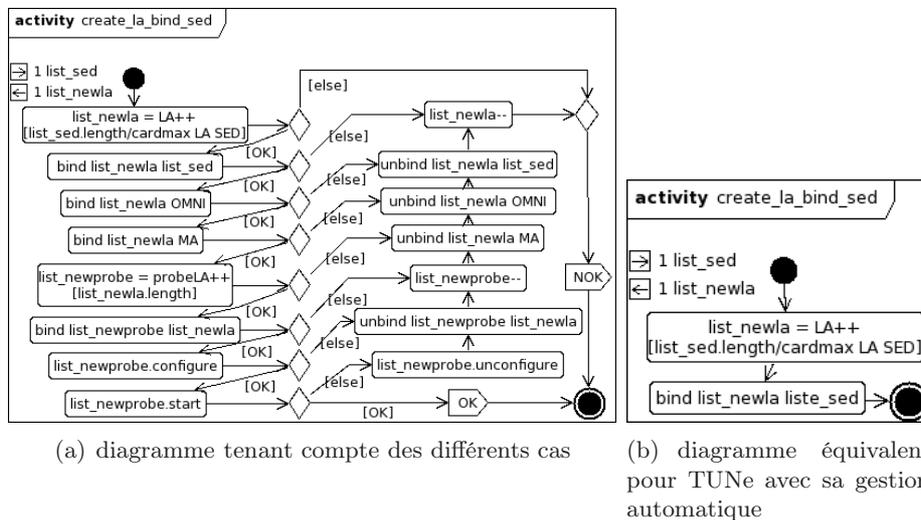


FIGURE 4.7 – PD pour l’optimisation de l’architecture de DIET



(a) diagramme tenant compte des différents cas

(b) diagramme équivalent pour TUNE avec sa gestion automatique

FIGURE 4.8 – Capacité des PD à s’autogénérer

L’utilisation conjointe des différents diagrammes permet de vérifier de manière statique et dynamique la cohérence du modèle. Cela offre aussi la possibilité de calculer des actions automatiquement sans que le programmeur n’ait besoin de les exprimer et donc de simplifier les PD (figure 4.8). Ceci s’appuie sur :

- des relations automatique entre les composants sans avoir besoin de l’exprimer explicitement (utilisation du SD)
- la création automatique de composant pour valider les diagrammes
- le "rollback" automatique si une action échoue et si le porgrammeur n’a pas défini la réaction à avoir en cas d’échec. Ceci permet de remettre le système dans un état cohérent.
- la propagation automatiquement des codes de retour pour les appels de PD.

#### 4.4.4 Quelques résultats

Cette expérience montre la capacité de TUNE à reconfigurer à chaud un logiciel comme DIET afin de l’adapter à la charge à laquelle, il doit faire face. En cas de surcharge, il crée de nouveau SED et en cas de souscharge il est capable de détruire des SED et de reconfigurer

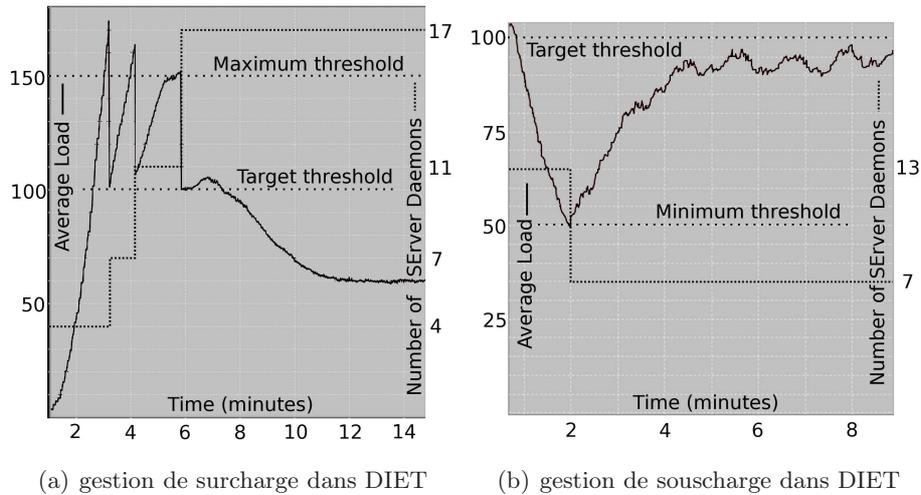


FIGURE 4.9 – Gestion de la capacité de traitement dans DIET

DIET. Dans les deux cas, on fonctionne avec des politiques à double seuils afin de ne pas avoir de phénomènes d’oscillations.

**Encadrement de thèse et publications :**

- Thèse de Rémi Sharrock
- R. Sharrock, T. Monteil, P. Stolf, D. Hagimont, L. Broto, Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements, International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS) vol.2 no.2, 2010

## 4.5 Bilan

Les travaux sur la couche intergiciel a nécessité beaucoup de conception et de développement en C, C++, JAVA. De nouveaux domaines scientifiques ont été abordés : les DSML et la méta-modélisation. Un dialogue actif avec le milieu industriel a été nécessaire afin de mieux capter les besoins des utilisateurs.

Le logiciel LANDA a été transféré à la société Delta Partners. Elle a assuré son industrialisation et sa commercialisation jusqu’à sa fermeture. Concernant l’environnement AROMA une demande de dépôt de brevet a été faite au CNRS en vu de son transfert à la société QoS Design lors de sa création. Suite à un recentrage des activités de cette startup et à un marché industriel difficile, ce transfert a été abandonné. Enfin, les travaux sur TUNe et plus particulièrement sur la gestion autonome de la performance permettent d’ouvrir de nombreux axes de recherche pour les années à venir dans le domaine de l’énergie ou bien encore dans l’exécution plus efficace des applications parallèles avec une coopération renforcée entre application et intergiciel.



# Chapitre 5

## La couche application

### 5.1 Introduction

Durant ces dernières années, on assiste à une convergence de l'architecture des supercalculateurs et des clusters. Ceci se retrouve au niveau des composants qui sont similaires et même au niveau du réseau de communication pour les clusters très performants. Toutefois, pour une bonne partie des clusters bâtis sur de l'éthernet ou pour les grilles, l'interconnexion reste un goulot d'étranglement. Ceci a pour conséquence que les applications de calcul tournant sur des clusters ou sur une grille devront soit échanger peu de données soit avoir un grain important (c'est à dire un rapport temps de calcul sur temps de communication grand). Deux classes d'applications sont donc de bonnes candidates pour la grille et les clusters :

- Les applications multi-paramétriques qui reviennent à lancer de nombreuses instances d'un même exécutable sans communication entre eux
- Les applications parallèles à gros grain dont les temps de calcul importants amortiront les temps de communication

Une première contribution concernera un code de simulation électromagnétique sur lequel nous avons appliqué des outils de déploiement autonome afin de lancer et contrôler un grand nombre d'instances dans le cadre d'une exécution multi-paramétrique. Sur les mêmes techniques de simulation, une pré-étude sur les gains potentiels obtenus par une parallélisation massive pour la simulation de structures sur-dimensionnées sera décrite.

Une deuxième contribution concernera l'analyse d'application parallèle à travers les traces de leur exécution et de leur code source. L'objectif est alors de corréliser trace et code source avec les entrées du programme afin de prédire le temps d'exécution et ainsi d'alimenter avec des valeurs plus fines les ordonnanceurs.

### 5.2 Code de simulation électromagnétique sur grille

#### 5.2.1 Contexte

Nous cherchons à créer des systèmes de plus en plus petits embarquant de plus en plus d'intelligence au niveau matériel et logiciel avec des architectures communicantes de plus en plus complexes. Ceci nécessite des méthodologies robustes de conception afin de réduire le cycle de développement et la phase de prototypage. Dans ce contexte, la conception et l'optimisation de la couche physique de communication est primordiale. La complexité de ces systèmes

rend difficile leur optimisation notamment à cause de l'explosion du nombre des paramètres inconnus. Les méthodes et outils développés ces dernières années seront à terme inadéquats pour traiter les problèmes qui nous attendent. Par ailleurs, les objets communicants seront très souvent intégrés dans des environnements encombrés de toutes sortes de structures métalliques et diélectriques de plus ou moins grandes tailles par rapport à la longueur d'onde. Le concepteur doit donc anticiper la présence de tels obstacles dans le canal de propagation afin d'établir des bilans de liaison corrects et un dimensionnement réaliste de l'objet communicant. Par exemple, la transmission d'onde dans un avion provenant de capteurs ou bien encore une antenne placée sur la jante d'une roue de véhicule et émettant des informations comme la pression d'un pneu en direction de l'habitacle voient ces caractéristiques de rayonnement grandement altérées par la présence de la structure métallique surdimensionnée que constitue la carlingue ou bien la carrosserie : il faut donc absolument tenir compte de cette perturbation pour prédire correctement les bilans de puissance entre l'antenne et un éventuel récepteur placé à l'intérieur de l'engin. Plus généralement, des avancées sur la partie théorique en électromagnétisme et en algorithmique sont nécessaires afin de proposer des outils informatiques pour le calcul rigoureux de la diffraction électromagnétique par des structures de très grandes dimensions ou du rayonnement d'antennes placées à proximité d'objets surdimensionnés. Ce calcul implique la résolution numérique de très grands systèmes difficilement accessible par les ressources informatiques traditionnelles. Une solution d'avenir consiste à s'appuyer sur des grilles de calcul. Yatpac<sup>1</sup> [95] est un logiciel de simulation électromagnétique basé sur la méthode de calcul TLM<sup>2</sup> [62]. Ce logiciel permet de caractériser un comportement électromagnétique dans un domaine temporel défini. Il peut simuler diverses structures. Ce logiciel est décomposé en plusieurs modules qui communiquent en s'échangeant des fichiers. Les principaux modules sont :

- YatGUI : est une interface graphique pour préparer les structures des simulations électromagnétiques.
- Yatpre : est un préprocesseur formattant les données des fichiers décrivant les structures à simuler en entrée pour les transformer en modèle TLM.
- Yati2of : est un prévisualisateur des structures sous la forme de modèle TLM.
- Yatsim : est le simulateur utilisant la méthode de calcul TLM.
- Yatspar : est un outil de post-processing permettant de sélectionner et formater les résultats d'une simulation pour sa future utilisation ou visualisation.
- Yatvis5d : est un outil de visualisation des résultats des simulations.

En fonction de la simulation à effectuer, les différents modules peuvent être lancés dans diverses configurations.

## 5.2.2 Déploiement et gestion des erreurs avec TUNe

Plusieurs codes de simulation électromagnétique ont été testés et exécutés sur la grille en utilisant TUNe. Ce travail est réalisé dans le cadre d'une collaboration forte avec des électroniciens où se mélange des problèmes de : traitement électromagnétique, algorithmique, parallélisme, informatique, simulation, etc. Dans le cas de Yatpac, l'objectif était de faire des simulations multi-run afin d'explorer un espace de solutions plus grand et de rendre plus robuste l'exécution des codes sur la grille en utilisant les capacités de TUNe. Ceci a permis de

---

1. Yet Another TLM Package  
2. Transmission Line Matrix

mieux cibler les limites et avantages de ce système. Des modules additionnels ont été conçus afin de :

- s'interfacer avec le gestionnaire de ressources OAR [23] utilisé sur GRID5000 [24].
- gérer des application multi-paramétriques
- collecter des informations de performance

La description du "flow" de la simulation Yatpac s'est fait simplement avec les diagrammes de description de l'application et les diagrammes d'état transition de TUNE :

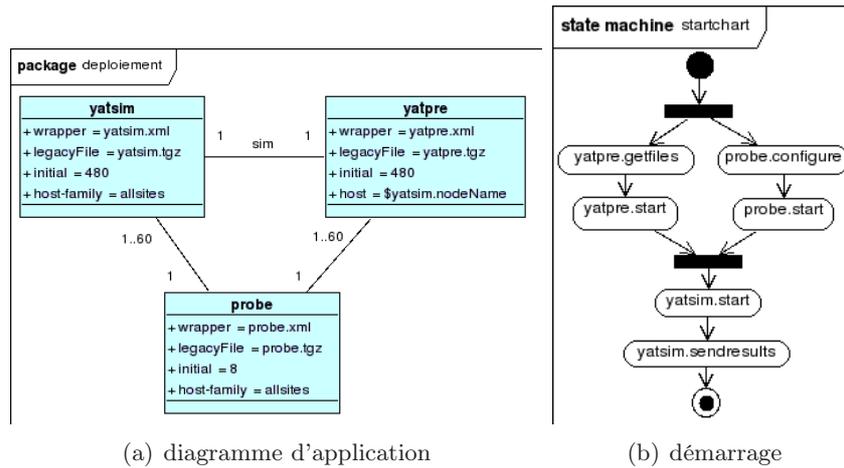


FIGURE 5.1 – Description en TUNE de Yatpac

Si TUNE détecte le non démarrage d'une des séquences nécessaires à l'exécution d'une des simulations, il est capable automatiquement de relancer la séquence en suivant le processus décrit via des diagrammes état transition.

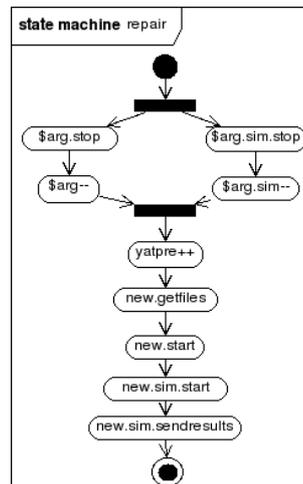
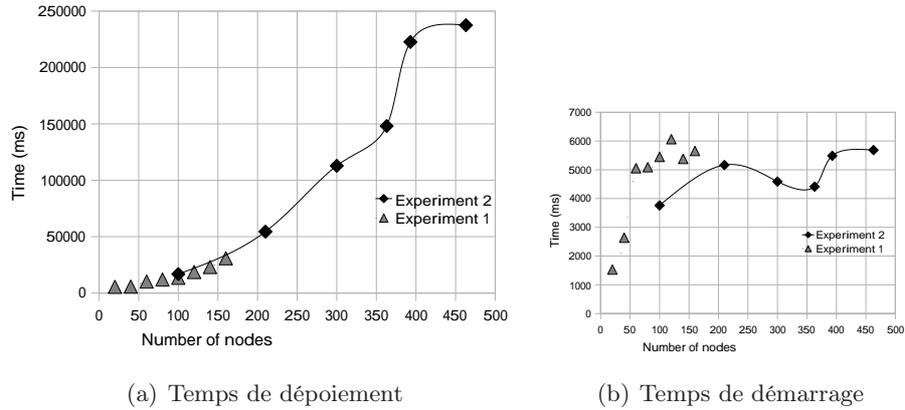


FIGURE 5.2 – Description de la réparation d'une simulation

Des expériences sur Grid'5000 en utilisant 6 sites différents ont été menées sur un nombre de noeuds de 120 (expérience 1) à plus de 460 noeuds (expérience 2). Des mesures sur le temps de déploiement des composants de Yatpac, de démarrage ou bien encore de réparation ont été réalisées (figure 5.3).



action	durée (ms)
nettoyage	683
destruction des composants	126
requête oagrid	4904
creation des composants	167
déploiement	4525
démarrage	1855

(c) réparation

FIGURE 5.3 – Quelques résultats de TUNe et Yatpac

Ceci a permis de mettre en évidence les limites de TUNe en terme de temps de déploiement et de gestion d'instance et à amener à réfléchir à de nouveaux schémas de fonctionnement interne pour TUNe.

#### Encadrement de thèse et publications :

- Thèse de Rémi Sharrock
- R. Sharrock, F. Khalil, T. Monteil, H. Aubert, F.Cocetti, P.Stolf, L. Broto, R. Plana, Deployment and management of large planar reflectarray antennas simulation on grid, Challenges of Large Applications in Distributed Environments (CLADE), Munich (Allemagne), 9-10 Juin 2009, 8p.
- F. Khalil, B. Miegemolle, T. Monteil, H. Aubert, F.Cocetti, R. Plana Simulation of micro electro-mechanical systems (MEMS) on grid, 8th International Meeting High Performance Computing for Computational Science (VECPAR'08), Toulouse (France), 24-27 Juin 2008, 14p.
- F. Khalil, E. B. Tchikaya, R. Sharrock, T. Monteil, F. Cocetti, H. Aubert, Grid-based SCT Approach for the Global Electromagnetic Simulation and Design of Finite-Size and Thick Dichroic Plat, ACES Journal (Applied Computational Electromagnetics Society, acceptée à paraître.

### 5.2.3 Pré-étude sur le passage sur grille à grande échelle

L'objectif est d'estimer d'une part les gains que l'on pourrait avoir en parallélisant la méthode TLM pour la grille et d'autre part de calculer les bonnes granularités entre calcul et communication. L'algorithme de la TLM est basé sur une discrétisation spatiale des structures

à simuler selon un schéma régulier en treillis ("Mesh"). L'algorithme de la TLM peut être vu de manière grossière ainsi :

```
for temps de simulation discrétisé do  
  for tous les points du maillage do  
    propager le champs à son voisinage  
  end for  
end for
```

On constate un couplage entre tous les points dû à la propagation des champs à tous les voisins d'un point.

Dans la suite, on suppose que le réseau est full duplexe / non bloquant (cas optimal). Ceci assure donc que les caractéristiques de débit et de latence du réseau extraits de l'article [51] sont généralisables dans le cas d'un grand nombre de communications simultanées. On suppose aussi qu'une tâche est capable de recevoir et d'émettre en même temps. Là aussi, on se place dans le meilleur des cas avec une programmation efficace de l'application. Ceci implique que les valeurs estimées dans la suite seront plutôt optimistes et que dans la réalité, on aura plutôt de moins bonnes valeurs, c'est un majorant qui est recherché. On suppose aussi que les processeurs de la grille sont libres et homogènes.

### 5.2.3.1 Estimation du temps de calcul en fonction du nombre de points

Nous cherchons d'une part à valider la vue simpliste de l'algorithme de la TLM et d'autres part à pouvoir extrapoler le temps que mettrait la simulation pour un nombre de points plus important. Vu l'algorithme, une modélisation avec une relation linéaire par rapport au nombre de point de discrétisation semble suffisante. On se base sur des exécutions du logiciel Yatpac pour extraire les coefficients avec la méthode des moindres carrés. Soit  $T_c(p)$  le temps de calcul d'un programme TLM avec  $p$  points de discrétisation, la relation suivante est obtenue avec une erreur relative inférieure à 1,7% (testée sur plusieurs exécutions réelles) :

$$T_c(p) = 0,000136527 * P + 15,03130435$$

### 5.2.3.2 Estimation de la taille en mémoire en fonction du nombre de points

De la même manière que précédemment, nous cherchons à donner une estimation de la taille en mémoire du programme en fonction du nombre de points de discrétisation  $R(p)$ . Nous supposons aussi que la taille mémoire du programme dépend linéairement du nombre de point de discrétisation. En utilisant de nouveau les exécutions de Yatpac et la méthode des moindres carrés sur une plage de mesure, nous déterminons la relation suivante avec des erreurs inférieures à 0,5% (testée sur plusieurs exécutions réelles) :

$$R(p) = p * 7,63 * 10^{-5} + 4,91$$

### 5.2.3.3 Estimation du gain en parallélisant la TLM par tranchage

On cherche à répartir le calcul en coupant en tranches la structure rectangulaire simulée. Un processeur calcul une tranche, envoi ensuite à ses voisins ses valeurs frontières et récupère

de ses voisins les valeurs frontières. Une tranche du milieu envoie et réceptionne de deux voisins ; une tranche d'extrémité envoie et réceptionne d'un seul voisin. Chaque calcul parallélisé sera réalisé par une tâche. Une tâche est affectée à un processeur.

La TLM est basée sur une boucle externe sur le temps. Le passage à l'itération suivante nécessite que les valeurs physiques aient été calculées sur toute la structure avant de passer au pas de simulation temporelle suivant. C'est une application synchrone. Ceci implique que l'application est cadencée par la tâche la plus lente c'est à dire celle qui travaille le plus. Dans notre cas, ce sera toutes les tâches modélisant des tranches du milieu. A chaque itération, ces tâches les plus lentes envoient 2 messages et réceptionnent simultanément 2 messages. soient :

- $I_t$  le nombre d'itérations sur le temps (dans l'exemple  $I_t = 3000$ )
- $x, y, z$  les dimensions de la structure et  $d_l$  le pas de discrétisation. On suppose que :  $x \geq y$  et  $y \geq z$  (dans l'exemple  $x = 4000, y = 4000, z = 100$ )
- $d_l$  le pas de discrétisation dans les trois directions (dans l'exemple  $d_l = 0,5$ )

Pour minimiser les échanges, il convient donc de « trancher » la structure selon l'axe  $y - z$ . Le nombre de points sur la frontière est donc :  $y/d_l * z/d_l$ . Un point est représenté à l'aide de douze valeurs codées par des réels en double précision. Dans la TLM, il échange avec chacun de ses voisins deux valeurs. La taille du message  $M$  échangé sur chaque frontière est donc en octets :  $y/d_l * z/d_l * 2 * 8$

Le réseau est modélisé par une fonction linéaire dépendant de la taille du message échangé :  $T_e = L + M/D$  où  $L$  est la latence,  $M$  la taille du message en octets et  $D$  le débit du réseau. Nous prendrons les valeurs mesurées sur GRID5000 dans [51] en négligeant les entêtes MPI. La performance d'une exécution parallèle est mesurée via le speedup :  $S(T)$  qui représente l'accélération de l'application en fonction du nombre de tâches  $T$ . Soit  $T_p(T)$  le temps de la tâche la plus longue quand on parallélise sur  $T$  tâches . On a :

$$S(T) = T_c(x/d_l * y/d_l * z/d_l) / T_p(T) \text{ avec } T_p(T) = T_c([x/d_l * y/d_l * z/d_l] / T) + (L + y/d_l * z/d_l * 2 * 8 * 2 / D) * I_t$$

Nous comparons le fonctionnement sur un cluster et une grille. Le speed-up est intéressant sur de très grosses structures avec plus 12800 millions de points (figure 5.4). Les estimations sur un cluster ou une grille se confondent à cause de la granularité choisie pour l'application. En terme de mémoire, cela devient possible à exécuter sur des machines de calcul à partir de

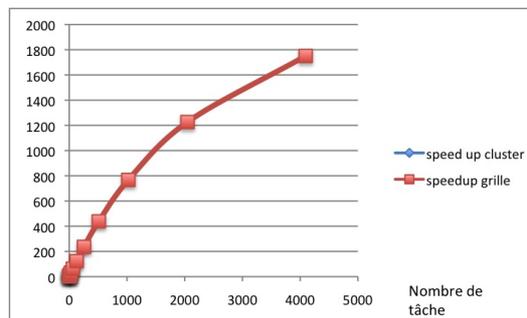


FIGURE 5.4 – Estimation du speed-up dans un découpage en tranches

32 tâches (30,56 Go), raisonnable pour 64 tâches (15,5 Go) et classique pour 256 tâches (3,8

Go).

#### 5.2.3.4 Estimation du gain en parallélisant la TLM par pavés

Dans la parallélisation précédente, quelque soit le nombre de tâches, la quantité de données échangée est la même et correspond aux points sur une tranche de la structure. L'idée est ici de décomposer la structure autrement. On décompose la structure à simuler en petites boîtes de même taille. On introduit un coefficient de décomposition  $C$ . Ce coefficient revient à construire des cubes de dimension  $x/C$ ,  $y/C$  et  $z/C$ . Ceci revient à créer  $C^3$  cubes. Là aussi le temps de l'application est calé sur le cube réalisant le plus d'échange c'est à dire les cubes à l'intérieur de la structure qui ont toutes les faces extérieures communes avec d'autres cubes. Quand le nombre de tâches augmente, on arrive à diminuer la quantité d'information échangée. On obtient donc un meilleur speedup.

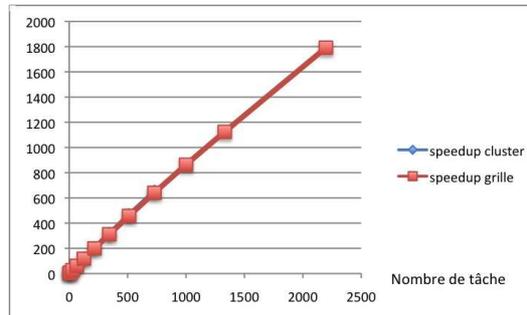


FIGURE 5.5 – Estimation du speed-up dans un découpage en pavés

#### Encadrement de thèse et publications :

- Thèse de Mihai Alexandru
- M. Alexandru, T. Monteil, F. Coccetti, P. Lorenz and H. Aubert, "Transmission-Line Modeling Computational Electromagnetics On Grids", Workshop on Present challenges in computational electromagnetics : complexity management, multi-scales, multi-physics, uncertainty management, statistics, St Malo 2-3 décembre 2010

## 5.3 Analyse de code

### 5.3.1 Contexte

De nombreuses recherches ont été effectuées dans le domaine de la prédiction du temps d'exécution d'applications, afin de déterminer comment relier le temps d'exécution d'une application avec le contexte dans lequel elle est lancée (entrées de l'application, plate-forme d'exécution, etc.). L'objectif final est ainsi de pouvoir estimer le temps d'exécution d'une application avant même qu'elle ne soit lancée. Dans le domaine du temps réel par exemple, l'utilité d'une telle donnée est cruciale au bon fonctionnement des systèmes, qu'ils soient critiques ou non [33]. En effet, les applications temps réel sont soumises à des contraintes temporelles (*deadlines*) qu'il convient de respecter impérativement pour les systèmes temps réel strict, ou au mieux pour les systèmes temps réel souple. Dans tous les cas, la connaissance du temps d'exécution des applications est nécessaire aux ordonnanceurs temps réel pour gérer l'ordre

d'exécution des applications qui leur sont soumises [6, 43, 17]. Plus qu'une simple estimation, ils utilisent le temps d'exécution des applications obtenu dans le pire des cas (*WCET* pour *Worst-Case Execution Time*). Les mécanismes d'ordonnancement appliqués aux domaines des grappes et des grilles de calcul nécessitent également de connaître, à priori, une estimation de la durée des applications à placer [72, 83, 81].

Jusqu'à présent dans nos travaux, nous avons considéré le programme à exécuter et ses paramètres comme une boîte noire. Dans cette approche, nous allons rentrer à l'intérieur du programme afin de l'analyser et de corréliser ses entrées à son temps d'exécution.

### 5.3.2 La problématique traitée

Nous nous intéresserons à des programmes séquentiels ou parallèles. Nous proposons une méthode hybride de prédiction de temps d'exécution qui combine plusieurs approches :

- une analyse sur l'historique des exécutions passées
- une analyse statistique sur les paramètres et fichiers d'entrées du programme
- une annotation du code source

### 5.3.3 Le modèle proposé

#### 5.3.3.1 Analyse des temps d'exécution passés

Nous définissons la notion de bloc de base comme un ensemble d'instructions élémentaires exécutées dans une seule et même fonction, et qui sont lancées le même nombre de fois, quelles que soient les entrées appliquées au programme. Ce nombre d'exécutions peut cependant varier d'une exécution à l'autre selon les entrées appliquées au programme. Le temps d'exécution d'un bloc de base sera considéré comme constant, c'est-à-dire indépendant du contexte d'exécution du programme. Cette hypothèse exclut la prise en compte des mécanismes présents sur les processeurs modernes, tels que les caches. En outre, le temps d'exécution des blocs de base est indépendant des entrées appliquées au programme, et ce en raison de l'absence d'instruction de branchement (notamment conditionnel) à l'intérieur des blocs de base. L'équation suivante peut être utilisée pour évaluer le temps d'exécution d'une application  $T_{App}(E)$ . A noter que celui-ci dépend des entrées du programme :

$$T_{App}(E) = \sum_{f \in \mathbb{F}} N_f^F(E) \cdot T_f^F(E), \text{ avec}$$

- $\mathbb{F}$  est l'ensemble des fonctions du programme,
- $N_f^F(E)$  est le nombre d'exécutions de la fonction  $f$ , dépendant des entrées  $E$ ,
- $T_f^F(E)$  est le temps d'exécution de la fonction  $f$ , dépendant également des entrées  $E$ .

Le temps d'exécution d'une fonction peut s'exprimer :

$$\forall f \in \mathbb{F} \quad T_f^F(E) = \sum_{b \in \mathbb{BB}_f^F} N_b^{BB}(E) \cdot T_b^{BB} \text{ avec}$$

- $\mathbb{BB}_f^F$  est l'ensemble des blocs de base de la fonction  $f$ ,
- $N_b^{BB}(E)$  est le nombre d'exécutions du bloc de base  $b$ , dépendant des entrées,
- $T_b^{BB}$  est le temps d'exécution du bloc de base  $b$ , considéré comme constant.

Le temps d'exécution d'une application peut donc s'écrire ainsi :

$$T_{App}(E) = \sum_{b \in \mathbb{BB}} N_b^{BB}(E) \cdot T_b^{BB}$$

avec  $BB$  ensemble des blocs de base du programme :  $\mathbb{BB} = \bigcup_{f \in \mathbb{F}} \mathbb{BB}_f^F$

L'utilisation des outils gprof ("profiler") et gcov (test de couverture) fournis par gnu permet de connaître pour une exécution donnée respectivement  $T_f^F$  et  $N_b^{BB}$ . Le temps d'exécution des blocs de base est la solution d'un système d'équations linéaires. Chaque exécution du programme pour différentes valeurs d'entrées permet d'ajouter une équation au système d'équations linéaires. Pour  $X$  exécutions, nous avons  $X$  équations linéaires et  $Card(\mathbb{B}\mathbb{B})$  inconnues. Nous supposons que le système possède suffisamment d'équations pour être résolu ( $X \geq Card(\mathbb{B}\mathbb{B})$ ); le système est donc potentiellement surdéterminé et est en fait mal conditionné. Nous avons donc reformuler le problème en introduisant une erreur et en passant à une résolution itérative [66]. Ceci a été validé sur un programme jouet de multiplication matriciel en exécutant réellement le programme et en testant la prédiction basée sur les précédentes exécutions (figure 5.6).

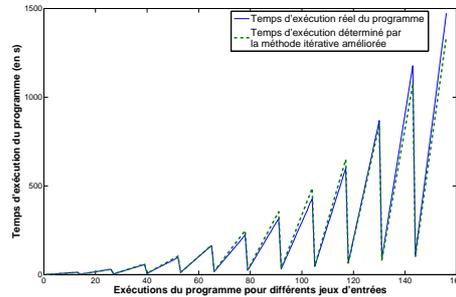


FIGURE 5.6 – Comparaison entre le temps d'exécution estimé et le temps d'exécution réel du programme

### 5.3.3.2 Corrélation entre les entrées et le temps d'exécution

L'objectif maintenant est d'estimer le nombre d'exécutions  $N_b^{BB}(E)$  de chaque bloc de base en fonction des entrées du programme contenue dans une base de données des précédentes exécutions (appelées aussi expériences) et de l'instance de l'application dont on souhaite prédire le temps d'exécution. Le modèle défini ci-dessous permet de sélectionner, au sein de la base de connaissances, un ensemble d'expériences semblables à la requête, puis de les combiner de manière à estimer le nombre d'exécutions des blocs de base d'un programme pour un vecteur d'entrées donné.

On note  $\mathbb{E}$  l'ensemble des vecteurs d'entrées admissibles du programme traité. Soit  $X$  le nombre d'expériences contenues dans la base de connaissances. Chaque expérience est associée à un vecteur d'entrées  $E^{(i)}$ , avec  $i \in [1; X]$ , de la forme :

$$E^{(i)} = \left\{ E_v^{(i)} \right\}_{v \in [1; N^V]} = \begin{bmatrix} E_1^{(i)} \\ E_2^{(i)} \\ \vdots \\ E_{N^V}^{(i)} \end{bmatrix} \quad \text{avec } E^{(i)} \in \mathbb{E}$$

où  $N^V$  correspond au nombre de valeurs constituant les entrées du programme. L'ensemble des vecteurs d'entrées correspondant aux expériences contenues dans la base de connaissances

est un sous-ensemble de  $\mathbb{E}$ , que nous notons  $\mathbb{E}^{Exp}$ . Afin de mesurer l'éloignement de deux contextes d'exécution, la distance euclidienne est utilisée [91]. Soient deux vecteurs d'entrées quelconques  $E^{(1)}$  et  $E^{(2)}$  du programme, celle-ci s'exprime de la manière suivante :

$$\forall E^{(1)} \in \mathbb{E} \quad , \quad \forall E^{(2)} \in \mathbb{E} \quad D(E^{(1)}, E^{(2)}) = \sqrt{\sum_{v=1}^{NV} d(E_v^{(1)}, E_v^{(2)})^2}$$

où  $d(E_v^{(1)}, E_v^{(2)})$  est la distance entre deux valeurs d'entrées définie à l'aide d'une métrique hétérogène de distance :

$$d(E_v^{(1)}, E_v^{(2)}) = \begin{cases} 1 & \text{si } E_v^{(1)} \text{ ou } E_v^{(2)} \text{ est inconnue,} \\ \text{si } E_v^{(x)} \text{ est nominale,} & \begin{cases} 0 & \text{si la valeur } E_v^{(1)} = E_v^{(2)}, \\ 1 & \text{dans le cas contraire.} \end{cases} \\ \text{si } E_v^{(x)} \text{ est linéaire,} & \frac{|E_v^{(1)} - E_v^{(2)}|}{\max_{E_v^{(x)}} - \min_{E_v^{(x)}}} \end{cases}$$

L'estimation du nombre d'occurrences des blocs de base dans le chemin d'exécution pour un vecteur d'entrées est réalisé grâce à une moyenne pondérée des valeurs contenues dans la base de connaissances. La pondération sera effectuée en fonction de la distance séparant les expériences de la requête, afin de favoriser l'influence des expériences les plus proches de la requête[81, 5]. Soit  $E^*$  le vecteur d'entrées correspondant à la requête, le nombre d'exécutions de chacun des blocs de base  $b$  du programme est estimé de la manière suivante :

$$\forall b \in \mathbb{BB} \quad N_b^{BB}(E^*) = \frac{\sum_{E \in \mathbb{E}^S} [K(D(E^*, E)) \cdot N_b^{BB}(E)]}{\sum_{E \in \mathbb{E}^S} K(D(E^*, E))}$$

où  $\mathbb{E}^S$  désigne un ensemble de vecteurs d'entrées correspondant aux expériences sélectionnées au sein de la base de connaissances. La fonction de pondération choisie est la fonction gaussienne :

$$K(d) = e^{-\left(\frac{d}{k}\right)^2}$$

avec  $k$  constante permettant de faire varier la largeur de la gaussienne. Cette constante permet de s'adapter à la densité d'expériences présentes dans la région contenant la requête. Pour valider l'approche, un programme de filtrage particulière parallélisé avec MPI a été utilisé. Il est bâti sur un modèle maître/esclave. La figure 5.7 montre une comparaison entre le nombre d'exécutions réel des blocs de base du programme pour un jeu d'entrées et celui estimé par le modèle de prédiction défini. Il est possible de constater que les deux courbes possèdent la même forme. Le modèle de prédiction parvient ainsi à déterminer quels sont les blocs dont le nombre d'exécutions varie en fonction des entrées, de même qu'à estimer les effets des valeurs d'entrées sur ces variations. L'erreur relative moyenne, constatée sur la prédiction de l'ensemble des blocs, s'élève à 8,8% pour le jeu d'entrées considéré.

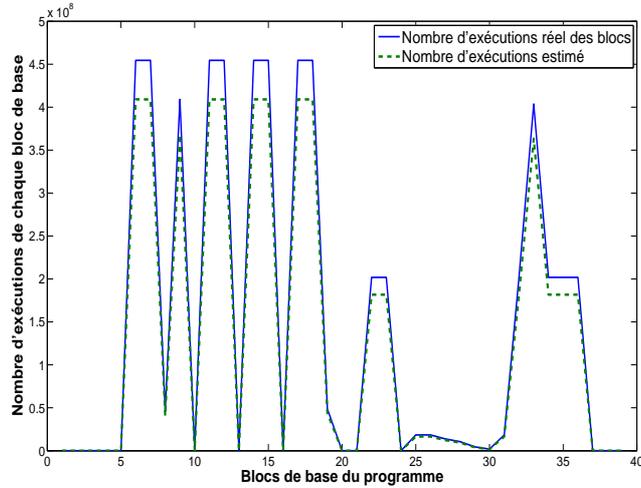


FIGURE 5.7 – Comparaison entre le nombre d'exécutions des blocs de base réel et celui estimé par le modèle de prédiction

### 5.3.3.3 Annotation des programmes

La formulation précédente suppose que toutes les valeurs d'entrées du programme traité ont une influence similaire sur le nombre d'exécutions de chacun des blocs de base du programme. Cette hypothèse est forte. Un des moyens de mieux prendre en compte l'impact de chaque entrée est d'ajouter un coefficient prenant en compte cela :  $w_{v,b}$ , dans le calcul de la distance pour l'entrée  $v$  et le bloc  $b$  :

$$\forall b \in \mathbb{BB} \quad , \quad \forall E^{(1)} \in \mathbb{E} \quad , \quad \forall E^{(2)} \in \mathbb{E} \quad D_b \left( E^{(1)}, E^{(2)} \right) = \sqrt{\sum_{v=1}^{NV} w_{v,b} \cdot d \left( E_v^{(1)}, E_v^{(2)} \right)^2}$$

Le choix des valeurs  $w_{v,b}$  dépend entièrement de la structure du programme. Ainsi, la personne la plus à même d'effectuer un tel choix est sans conteste le développeur de l'application. Les annotations sont ainsi un moyen souple et facile de permettre au programmeur de renseigner ces valeurs. Elles doivent répondre à ces caractéristiques :

- permettre à l'utilisateur de donner la dépendance du nombre d'exécutions des blocs vis-à-vis des entrées du programme,
- ne nécessiter aucune connaissance du modèle mathématique de prédiction de comportement d'applications,
- être insérables directement dans le code source du programme,
- ne pas empêcher la compilation ni le bon fonctionnement en exécution de l'application.

Nous avons choisi dans le cadre du C, l'utilisation des directives `#pragma` qui sont ignorées par le compilateur avec la syntaxe suivante : `#pragma etp annotation (paramètres)` où :

- `etp` signifie *execution time prediction*. Ce mot-clé caractérise donc l'ensemble des annotations que nous définissons, de manière à ce qu'elles soient immédiatement identifiables par l'outil mis en place pour les interpréter.
- `annotation` désigne le type d'annotations. Il existe quatre types différents d'annotations, décrits ci-dessous.

- (*paramètres*) est une liste de paramètres facultatifs, séparés par des virgules, et placés entre parenthèses.

Quatre types d’annotations sont définis :

- **Annotations de type “inputs”** : Ce type d’annotations permet de définir les entrées considérées dans la prédiction du temps d’exécution. De telles annotations peuvent être insérées au début du code source, au côté des traditionnelles directives `#define`.
- **Annotations de type “begin dependency”** : Ce type d’annotations débute une séquence d’instructions dont le nombre d’exécutions dépend d’entrées du programme. La liste de ces entrées, telles que définies par la directive “inputs”, est spécifiée en paramètres.
- **Annotations de type “begin independency”** : Ce type d’annotations débute une séquence d’instructions dont le nombre d’exécutions ne dépend d’aucune entrée du programme.
- **Annotations de type “end”** : Ce type d’annotations clôt toute séquence d’instructions, faisant ainsi suite aux annotations de type “begin dependency” ou “begin independency”.

Par exemple, la ligne suivante peut être ajoutée au programme de filtrage particulière, dans le but de définir les trois entrées considérées :

```
#pragma etp inputs (particules, intervalles_temps, taches_esclaves) puis :
#pragma etp begin dependency (particules)
for (int i = 0; i < particules; i++) {
    instructions_1;
    #pragma etp begin dependency (intervalles_temps)
    for (int j = 0; j < intervalles_temps; j++) {
        instructions_2;
    }
    #pragma etp end
    instructions_3;
}
#pragma etp end
```

Ainsi, les blocs d’instructions 1 et 3 ne dépendent que du nombre de particules, tandis que le bloc d’instructions 2 dépend à la fois du nombre de particules et du nombre d’intervalles de temps.

Les mêmes expériences que précédemment ont été réalisées et l’erreur de prédiction avec ou sans annotation est comparée sur la figure 5.8 qui montre l’amélioration sensible de la prédiction en utilisant les annotations.

#### **Encadrement de thèse et publications :**

- Thèse de Bernard Miégemolle
- B. Miegemolle, T. Monteil, Hybrid method to predict execution time of parallel applications, The 2008 International Conference on Scientific Computing (CSC’08), Las Vegas (USA), 14-17 Juillet 2008, 7p.

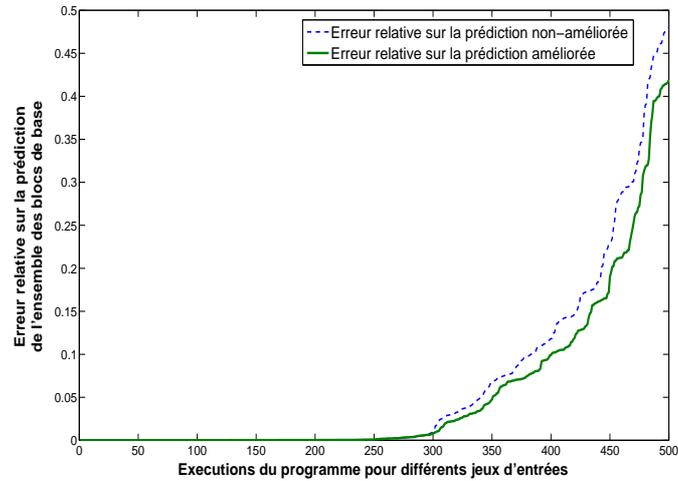


FIGURE 5.8 – Comparaison des erreurs obtenues sans et avec annotations

## 5.4 Bilan

Le travail sur la couche application a permis d'aborder des techniques et domaines différents comme : la simulation électromagnétique basée sur la TLM, l'analyse de code, les outils de "profiling", des méthodes d'analyse numérique et de calcul matriciel.

Les premiers résultats sur la parallélisation de la TLM ont montré qu'il fallait réduire au maximum l'utilisation du réseau pour être efficace et simuler de grosses structures par rapport à ce qu'il se fait traditionnellement. L'ordre de grandeur passe à des structures de plusieurs mètres, ce qui correspond aux défis auxquels les électroniciens font ou vont avoir à faire face. Cette pré-étude a donné lieu à la création d'une thèse en cours que je co-encadre avec un électronicien et l'inscription d'un challenge dans le projet INRIA HEMERA.

Concernant, la prédiction du temps d'exécution des applications, une méthode originale a été conçue. Elle sera utilisée par les nouvelles applications parallèles autour de la simulation électromagnétique sur lesquelles je vais travailler afin d'alimenter les ordonnanceurs et les environnements autonomiques qui contrôleront l'application parallèle.



# Chapitre 6

## Synthèse et perspectives

### 6.1 Résumé des contributions

L'ensemble des travaux de recherche effectués peut être vu sous l'angle de la recherche de performance au sens large. Cette dernière a été explorée selon différentes facettes.

*La recherche de performance pour l'application parallèle* constitue un premier axe, ceci se traduit par des contributions dans le calcul et la minimisation du temps d'exécution ou du temps de restitution des résultats, plus précisément :

- La simulation événementielle et analytique d'applications MPI s'exécutant sur une plateforme distribuée avec une modélisation fine du matériel (carte réseau, mémoire, disque), du protocole de communication, du système d'exploitation (Linux) et des intergiciels (LAM-MPI) afin de prédire le temps d'exécution pour gérer ou planifier des plateformes distribuées par exemple (paragraphe 2.2)
- La prise en compte des incertitudes sur la connaissance de la charge d'une machine multi-processeur par une modélisation stochastique puis la prédiction du temps d'exécution d'un programme parfaitement connu venant s'exécuter sur ce système aléatoire (paragraphe 2.3)
- la modélisation de la localisation des messages dans les environnements parallèles de communication par passage de messages et la mise en place de politiques de routage dynamique des messages afin d'améliorer le temps de calcul (paragraphe 2.4)
- La proposition d'un algorithme de placement créant des classes de service permettant de prendre en compte différents types d'utilisateurs et leurs contraintes : demande de résultat avant une certaine date, exécution immédiate, etc (paragraphe 3.2.2)
- Une sorte de méta-ordonnanceur prenant des décisions au niveau de la grille qui répartit les demandes de placement sur les ordonnanceurs locaux de chaque site ou cluster tout en prenant en compte les temps réseaux dans le cas de la distribution d'une application sur plusieurs sites géographiques (paragraphe 3.2.3)
- Le déploiement et l'exécution d'applications parallèles comme Yatpac en tenant compte des éventuelles défaillances par des politiques autonomiques afin d'optimiser le temps de restitution des résultats (paragraphe 5.2.2)
- L'étude de la parallélisation massive de simulations électromagnétiques par le méthode TLM afin d'une part de traiter des structures sur-dimensionnées et d'autre part d'obtenir un temps d'exécution raisonnable (paragraphe 5.2.3)

- La prédiction du temps d'exécution par l'analyse de trace et du code source de programmes parallèles afin de fournir aux ordonnanceurs une information la plus réaliste possible sur le temps de processeur nécessaire en fonction des entrées de l'application (paragraphe 5.3).

*Des premiers travaux ont été réalisés sur la recherche de la performance énergétique* sur les coeurs de réseau des plates-formes de calcul tout en garantissant un niveau de qualité de service en terme de débit. Le problème a tenu compte d'applications pouvant demander différentes valeurs de qualité de service et a permis de trouver la configuration des routeurs (paragraphe 2.5).

*La recherche de la performance financière* en terme de coût d'utilisation dans le cas de facturation des ressources fluctuant avec le temps a été étudiée. Cela a permis de fournir un algorithme d'ordonnement offrant un compromis entre temps d'exécution et coût (paragraphe 3.2.4).

*Enfin la performance d'utilisation* des clusters et grille en terme de facilité d'accès et de manipulation a été abordée avec la proposition ou la participation à divers environnements :

- Aroma constitue un environnement complet pour la mise en place d'une structure ASP, il fournit des outils d'observation et de statistique sur une grille, la gestion des utilisateurs et l'accès aux logiciels en mode ASP (paragraphe 3.3 et 4.3)
- LANDA permet d'exécuter très facilement une application parallèle sur un cluster, il a servi entre autres pendant des années à un cours sur la programmation parallèle qui permettait en quelques heures à des étudiants d'écrire une application parallèle, de l'exécuter et d'observer son comportement (paragraphe 4.2)
- TUNe a été étoffé par des politiques de reconfiguration dynamique qui permettent par exemple d'adapter la performance d'un logiciel durant son exécution (paragraphe 4.4).

Une autre manière de considérer, l'ensemble des recherches présentées dans ce manuscrit est de prendre comme thématique centrale la gestion de ressources dans un système distribué (figure 6.1). Tous les travaux décrits viennent alors enrichir ce thème principal :

- Les algorithmes de placement, de prédiction et de gestion (paragraphe : 2.3, 3.2.2, 3.2.3, 5.3, 2.5, 3.2.4 et 4.4)
- Le développement et l'analyse des applications parallèles permet de maîtriser ces dernières et ainsi d'avoir un ensemble d'applications tests (paragraphe : 5.2.2 et 5.2.3).
- Le développement d'environnements (paragraphe : 2.4, 3.3, 4.3 et 4.2) : TUNe, LANDA et AROMA permet de développer, déployer et contrôler les applications sur des ressources réelles et ainsi de disposer de tout ce qui est nécessaire pour valider réellement les algorithmes de gestion de ressources.
- L'intérêt pour les simulateurs (paragraphe : 2.2) est de pouvoir valider soit plus rapidement des algorithmes de gestion soit des choses que l'on ne peut valider en réel (par exemple pour des problèmes de limitation du nombre de ressources)

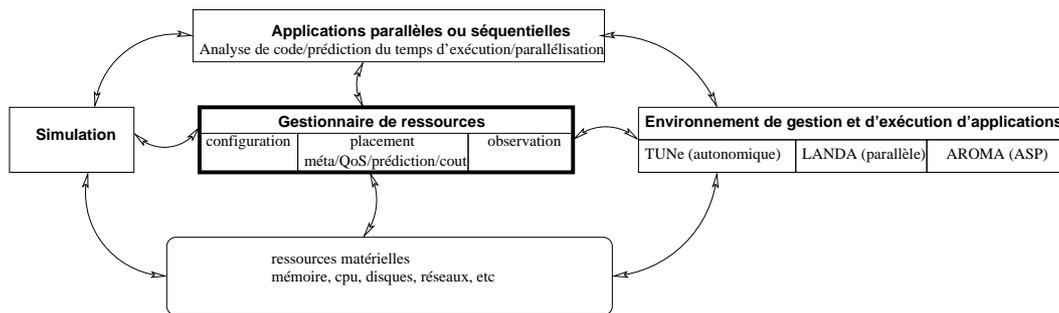


FIGURE 6.1 – Synthèse des travaux

## 6.2 Vision et Perspectives de recherche

### 6.2.1 Evolution des ressources matérielles

Si on s'intéresse à l'évolution des architectures des ressources informatiques pour le calcul sur ces vingt dernières années. On observe le passage peu à peu d'architectures très spécifiques de calculateurs intégrés basées sur des processeurs spécialisés à des architectures beaucoup plus modulaires et basées sur des composants génériques. Dans les années 80-90 des supercalculateurs représentant bien cette tendance sont le cray X-MP et ses processeurs vectoriels, le cray T3D ou bien l'origin 2000 et leurs processeurs scalaires. Parallèlement à la même époque, on observe une montée en puissance et une augmentation des stations de travail. Ceci ouvre la voie à un nouveau support pour le calcul parallèle avec la création de machine parallèle virtuelle basée sur des couches logicielles et utilisant les processeurs interconnectés par un réseaux de communication non dédié. C'est la création des logiciels PVM [46] ou bien LANDA [70]. L'aspect économique et le besoin en calcul a poussé à l'émergence des grappes (ou cluster), datacenter ou grilles de calcul bâtis sur le modèle de machine parallèle virtuelle. La crise autour des sociétés développant les supercalculateurs, la progression des technologies réseau et le développement des couches logiciels nous amènent de nos jours à retrouver à l'intérieur d'un supercalculateur une architecture très proche d'un cluster ou d'une grille. Par exemple, le nouveau calculateur universitaire Toulousain (groupement CALMIP [21] ) est constitué de différents éléments : noeuds simples de calcul, noeuds SMP, noeuds de cartes graphiques (pour la visualisation ou le calcul) interconnectés par le même réseau (figure 6.2).

Cette architecture se retrouve aussi dans les "datacenters" où l'on ne parle plus de calcul mais de service en général. Elle se retrouve aussi au niveau d'une grille si on fait l'abstraction de l'architecture réseau longue distance. Il semble que tant qu'il n'y aura pas un bond technologique très fort (par exemple la percée des ordinateurs quantiques prévu pour 2020 [34]), nous resterons sur des architectures très proches pour le monde du calcul et le monde du service.

Ceci à une incidence directe sur mes travaux de recherche futurs. En effet, les modélisations, algorithmes de gestion, observation des ressources développés pour le calcul trouvent un nouveau champ d'application dans le monde de l'accès aux services distants en général. Ceci expliquera les projets de recherche explicités plus loin qui concernent à la fois le monde du calcul mais aussi le monde du service.

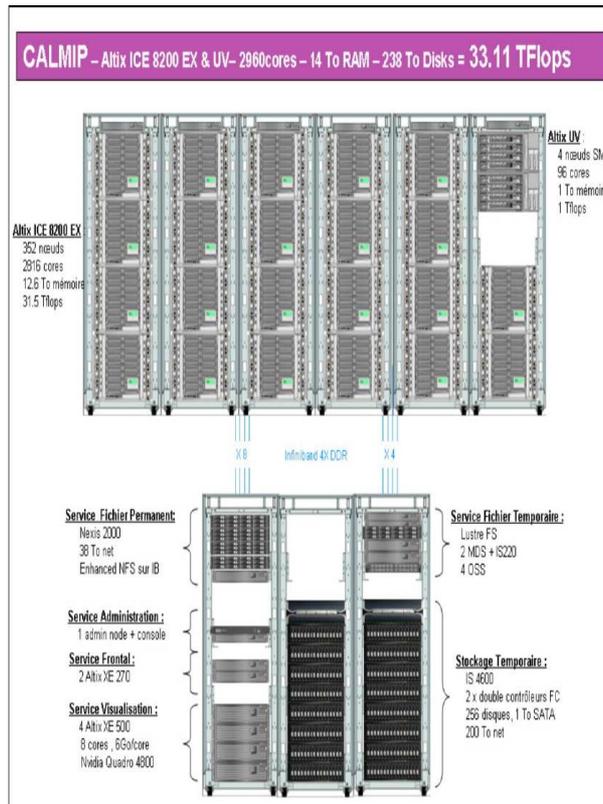


FIGURE 6.2 – Exemple d’architecture d’un méso-centre de calcul

## 6.2.2 Evolutions des pratiques et des mentalités

En 1991, quand j’étais stagiaire on accédait à un ordinateur soit grâce à un terminal de type VT100 connecté à un petit serveur soit on disposait d’une station de travail pour les plus chanceux. On retrouvait donc deux types de pratiques bien distinctes : soit une station dédié soit l’accès à une ressource partagée entre plusieurs utilisateurs. Jusqu’au début des années 2000 l’informatique s’est surtout développée sur la première stratégie avec l’explosion de l’informatique personnelle et l’avènement des ordinateurs portables embarquant tous les logiciels et données nécessaires à leur fonctionnement. Depuis le début des années 2000, on rebascule vers une informatique plus mixte avec à la fois une utilisation locale et une montée de plus en plus forte d’accès à des ressources et services distants. Les produits développés par "google" en sont une illustration (googledoc, gmail, etc). Cette évolution vers toujours plus d’accès distant est pilotée par différents leviers : la facilité de communication via des standards, la rapidité grandissante du réseau, le besoin en mobilité géographique, le besoin d’accéder à ses données voir à ses applications depuis des terminaux différents (ordinateur portable, smartphone, tablette, etc). On estime à une moyenne de 6 terminaux par personne connectés à Internet dans les années à venir, et bien sûr les utilisateurs voudront accéder à des choses communes depuis ces différents équipements.

Cette explosion de l’information dans notre société et de ses usages amène une population d’utilisateurs de l’informatique très variée. Pour un individu, ne pas suivre ce train technolo-

gique revient à se mettre en marge de la société à terme.

Enfin depuis quelques années une préoccupation majeure dans nos activités doit être prise en compte : il s'agit de l'impact écologique et énergétique des activités autour de l'internet et des technologies de l'information. Le matériel commence à prendre en compte ces aspects mais nous restons encore dans un modèle de sur-consommation énergétique. La difficulté résultant dans la complexité d'extraire des politiques de gestion dans le cocktail formé par les utilisateurs, le matériel et les logiciels.

Il existe de nombreux défis à relever, nous pouvons citer :

- la nécessité d'une transparence totale sur la complexité des architectures mise en place et l'accès à l'information et aux services
- le besoin de mettre en place des gestionnaires de ressources intégrant de multiples critères : coût, QoS, énergie, localisation, etc
- la structuration de l'accès aux multitudes de services qui s'offriront à nous
- l'extensibilité des solutions sur des échelles jamais atteintes
- la sécurité de tel système
- la caractérisation des utilisateurs très variés
- la modélisation des services complexes
- etc

La transparence, la modélisation des services et des utilisateurs ainsi que la prise en compte de multiples critères dans la gestion des ressources que ce soit pour le calcul ou pour l'accès à des services constitueront un autre fil conducteur de mes activités de recherche à venir.

### 6.2.3 Evolution de la complexité

Une mesure de la complexité qui nous attends au niveau de l'informatique peut être donné par l'évolution de la quantité d'information que l'on prévoit sur internet, selon le département d'information et de gestion des systèmes de l'Université de Berkeley, d'ici 2 ans, la quantité d'information sur internet doublera tous les 11 heures. Les applications gérant cela devront donc être capables de s'adapter en continue. Elles seront bâties sur des architectures complexes pour s'interfacer avec les différentes sources d'information mais aussi pour faire face aux utilisateurs. La gestion humaine de ces systèmes pose déjà un problème de nos jours mais sera inimaginable pour des problèmes de temps de réponse et de qualité des réponses apportées en cas de problème car le système dans son ensemble sera impossible à appréhender.

La gestion autonome est une des solutions. Nous en sommes à ces débuts. Il reste de nombreuses choses à développer : l'intégration dans le cycle de développement des applications, la généricité des solutions apportées, les formalismes d'expression, la mise à l'échelle, etc. Nous essayerons d'apporter des contributions sur certains de ces aspects dans les années à venir.

### 6.2.4 Quelques projets à venir

Les exemples des trois projets ci-dessous mettent en avant les thématiques sur lesquelles je mise sur les années à venir avec en arrière plan toujours une préoccupation sur l'utilisation des ressources distribuées. Le monde informatique que nous construisons sera basé sur une informatique de services transparents s'appuyant sur des énormes centres de calcul/données répartis dans le monde qu'il faudra gérer au mieux par des algorithmes décentralisés de gestion

dans des outils autonomiques. Le monde du calcul suivra aussi cette tendance.

Le premier projet concerne la simulation électromagnétique. Depuis un an, dans le cadre d'un co-encadrement d'une thèse, nous travaillons sur la partie théorique du simulateur en coopération avec des électroniciens. Ceci a permis d'aboutir à un premier programme séquentiel pouvant coupler plusieurs méthodes de simulation. La partie parallèle a commencé et une première version basée sur MPI donne de premiers résultats. Mes activités vont tourner autour d'une parallélisation la plus efficace possible (basée sur le multi-threading et MPI) et surtout sur la connexion à un environnement autonome à étendre afin de piloter les simulations. Les objectifs sont multiples :

- assurer que l'application s'exécute correctement dans le cas de l'utilisation de plusieurs milliers de nœuds de calcul
- passer à un ordre de grandeur jamais atteint pour ces techniques avec la simulation de plusieurs milliards de nœuds de discrétisation électromagnétiques
- permettre à la simulation de s'étendre ou de se contracter en fonction des ressources disponibles automatiquement
- insérer dans le simulateur de l'intelligence afin qu'il puisse automatiquement lancer d'autres simulations afin d'explorer de meilleures solutions (une sorte de multi-run piloté automatiquement) en s'appuyant sur des politiques autonomiques.

Le second concerne la coopération entreprise avec l'IRIT sur le domaine de l'économie d'énergie. Ce projet a pour objectif de développer des modèles mathématiques de la consommation énergétique dans les centres de données/calcul et de concevoir des stratégies permettant d'adapter dynamiquement la consommation énergétique des centres de données à la demande applicative en fonction des besoins en qualité de service. Ceci passera par différentes étapes :

- Modélisation globale de la consommation énergétique par :
  - Caractérisation de la plate-forme d'exécution
  - Caractérisation des applications et de leurs besoins en qualité de service
  - Création de Modèle globale de la consommation énergétique et des performances applicatives
- Gestion adaptative de la consommation énergétique par :
  - Mesures et actions d'économie d'énergie
  - Algorithme de décision centralisé et distribué
- Expérimentation réel dans un centre de données/calcul

Enfin la proposition d'un projet sur les 3 années à venir sur de nouveaux modèles de fonctionnement pour l'informatique des particuliers sera faite. Ce projet part de la constatation que les téléphones mobiles et les ordinateurs personnels se rapprochent de plus en plus. Toutefois, d'un coté le modèle d'accès aux services/applications sur les téléphones mobiles a été conçu pour faciliter l'accès aux utilisateurs novices avec des modèles économiques de gratuité, de facturation à l'achat ou à l'utilisation. De l'autre, pour les ordinateurs, nous sommes encore très largement sur un modèle de fonctionnement où l'utilisateur doit gérer sa machine en terme de système d'exploitation, acheter ses logiciels, les installer et les mettre à jour avec les conséquences que cela peut avoir pour un utilisateur novice. Il me semble que ceci est propice à l'émergence d'un nouveau modèle de fonctionnement de l'informatique qui doit s'appuyer sur les acquis de la téléphonie mobile en terme de service et d'usage, de l'extension des modèles

d'accès distant aux ressources logicielles (SAAS, cloud computing), des apports de l'informatique autonome pour la réactivité et du développement des machines virtuelles. L'objectif de ce projet serait de construire un modèle de fonctionnement hybride pour l'informatique mélangeant les avantages des modèles d'installation/gestion locales des machines et des logiciels ainsi que l'accès à des services distants dans des "datas center" ou sur la machine d'autres utilisateurs dans un mode de vol de cycle dans un esprit "green". Le choix du type de fonctionnement sera piloté par des modèles en fonction de l'utilisateur, de la machine (puissance trop faible), du centre de ressources (charge, consommation, licence), des logiciels (non accessible à distance, trop consommateur de bande passante, mutualisable), de la qualité de service souhaitée (rapidité, confidentialité, redondance), de l'énergie consommée, etc. Non seulement il faudra travailler sur la modélisation et la mise en oeuvre d'un tel système mais aussi sur sa simulation afin de montrer la faisabilité de cette démarche pour plusieurs centaines de milliers voire de millions d'utilisateurs.



# Bibliographie

- [1] A. Abbas. *Grid Computing : A Practical Guide to Technology and Applications*. Networking Series, 2003.
- [2] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Journal of Future Generation Computer Systems (FGCS)*, Octobre 2002.
- [3] J. Aman, C. Eilert, D. Emmes, P. Yocom, and D. Dillenberg. Adaptive algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2) :242–283, 1997.
- [4] D. Artiges. *Contrôle et évaluation de réseaux de télécommunication*. PhD thesis, Université de Nice, 1996.
- [5] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, Vol. 11, Février 1997.
- [6] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Fixed priority pre-emptive scheduling : An historical perspective. *Real-Time Systems*, Vol. 8, 1995.
- [7] David A. Bacigalupo, Stephen A. Jarvis, Ligang He, Daniel P. Spooner, Donna N. Dillenberg, and Graham R. Nudd. An investigation into the application of different performance prediction methods to distributed enterprise applications. *The Journal of Supercomputing*, 34(2) :93–111, 2005.
- [8] S. Basu, V. Talwar, B. Agarwalla, and R. Kumar. Interactive grid architecture for application service providers. In *First International Conference on Web Services, Las Vegas (USA)*, Juin 2003.
- [9] K.M. Baumgartner and B.W. Wah. Computer scheduling algorithms : past, present and future. *Information Sciences*, vol 57 et 58, pp319-345, Elsevier Science Pub. Co., New York, NY, Sept-Dec 1991.
- [10] D. S. Bedassé, T. Huang, M. A. Munawar, and J. J. Wu. Event-based self-management. In *Management, 2008*, 04–02.
- [11] G. Bernard and B. Folliot. Caractéristiques générales du placement dynamique : synthèse et problématique. In *Ecole placement dynamique et répartition de charge : application aux systèmes parallèles et répartis, Presqu'île de Giens*, pp115-124, Juillet 1996.
- [12] D.P. Bertsekas. *Dynamic programming, Deterministic and Stochastic Models*. Prentice-hall., 1987.
- [13] L. Broto, D. Hagimont, P. Stolf, N. de Palma, and S. Temate. Autonomic management policy specification in tune. In *ACM symposium on Applied computing*, 2008.

- [14] Bruneton, Coupaye E., Leclercq T., M., V. Quéma, and J. Stefani. The fractal component model and its support in java. *Software : Practice and Experience*, 36(11-12), 1257-1284.
- [15] Greg Burns, Raja Daoud, and James Vaigl. LAM : An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.
- [16] R.M. Butler and E.M. Lusk. Monitors, messages and clusters : the p4 parallel programming system. *Parallel Computing (20)*, pages 547–564, 1994.
- [17] G. C. Buttazzo. *Hard-Real Time Computing Systems : Predictable Scheduling Algorithms and Applications*. Springer, Seconde Edition, Octobre 2004.
- [18] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service-oriented grid computing. In *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001, San Francisco (USA), Avril 2001*.
- [19] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *The Journal of Concurrency and Computation : Practice Experience (CCPE), Wiley Press, Vol. 14, 2002*.
- [20] R. Calkin, R. Hempel, H-C. Hoppe, and P. Wypior. Portable programming with the parmacs message passing library. *Parallel Computing (20)*, pp. 615-932, 1994.
- [21] Calmip homepage, <http://www.calmip.cict.fr>.
- [22] C.H Cap and V. Strumpfen. Efficient parallel computing in distributed workstation environments. *Parallel Computing (19)*, pp. 1221-1234, 1993.
- [23] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard. A batch scheduler with high level components. *Cluster computing and Grid 2005 (CCGrid05), Cardiff (Grande-Bretagne), 9-12 Mai 2005*.
- [24] F. Cappello, E. Caron, Dayde M., Desprez F., Y. Jegou, P. Primet, and Jeannot. Grid'5000 : a large scale and highly reconfigurable grid experimental testbed. In *6th IEEE/ACM International Workshop on Grid Computing, 99–106. Seattle, Washington, USA*.
- [25] E. Caron and F. Desprez. Diet : A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3) :335-352.
- [26] N.J. Carriero, D. Gelernter, T.G. Mattson, and A.H. Sherman. The linda alternative to message passing systems. *Parallel Computing (20)*, pages 633–655, 1994.
- [27] H. Casanova. Simgrid : A toolkit for the simulation of application scheduling, <http://simgrid.gforge.inria.fr/>.
- [28] T.L. Casavant and J.G. Kuhl. A taxinomy of scheduling in general-purpose distributed computing system. *IEEE Transactions on software engineering*, vol 14, N 12, February 1988.
- [29] Site cplex, <http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio>.
- [30] M. Daydé. Grid'mip, <http://gridmip.enseeiht.fr/index-french.html>.
- [31] P.J. Denning. *Operating Systems Theory*. Prentice Hall, 1990.
- [32] Article sur le site web de enerzine, <http://www.enerzine.com/603/6982+la-consommation-des-datacenters-un-enjeu-planetaire>.

- [33] A. Ermedahl, F. Stappert, and J. Engblom. Clustered worst-case execution-time calculation. *IEEE Transactions on Computers*, Vol. 54, Septembre 2005.
- [34] D. Evans. Top 25 technology predictions. Technical report, Cisco IBSG Innovations Practice.
- [35] P. Farail, P. Gauffillet, A. Canals, C. Le Camus, Sciamma, P. D., Michel, and X. Crégut. The topcased project : a toolkit in open source for critical aeronautic systems design. In *European Congress on Embedded Real Time Software*, 54-59. Toulouse, France.
- [36] D.G. Feitelson. <http://joblog.npaci.edu/>.
- [37] D.G. Feitelson. <http://www.cs.huji.ac.il/labs/parallel/workload/>.
- [38] D.G. Feitelson. <http://www.cs.huji.ac.il/labs/parallel/workload/logs.html#sdscsp2>.
- [39] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. Theory and practice in parallel job scheduling. In *Proceedings workshop Job Scheduling Strategies for Parallel Processing*, Geneva, 1997.
- [40] D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. *Economic Models for Allocating Resources in Computer Systems*. Market-based Control : A Paradigm for Distributed Resource Allocation, World Scientific Press, Singapore, 1996.
- [41] J. Flower and A. Kolawa. Current and future directions in express. Technical report, Parasoft Corporation.
- [42] Message Passing Interface Forum. Mpi : A message-passing interface standard. *International Journal of Supercomputing Applications*, 8(3/4), 1994.
- [43] J. Ganssle. Really real-time systems. In *Embedded Systems Conference (ESC SF)*, Avril 2001.
- [44] J.M. Garcia. processus de décision markovien, application à l'acheminement d'appels téléphoniques. Technical report, LAAS, 1983.
- [45] David Gauchard. *Simulation hybride des réseaux IP-DIFFSERV-MPLS multiservices sur environnement d'exécution distribuée*. PhD thesis, Institut National des Sciences Appliquées de Toulouse, 2004.
- [46] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Bob Manchek, and Vaidy Sunderam. *PVM : Parallel Virtual Machine - A User's Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.
- [47] G.Franks, A. Hubbard, S. Majumdar, D.C. Petriu, J. Rolia, and C.M. Woodside. A toolset for performance engineering and software design of client-server systems. *Performance Evaluation*, 24(1-2) :117–135, 1995.
- [48] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.
- [49] M. Goldszmidt, D. Palma, and B. Sabata. On the quantification of e-business capacity. In *ACM Conference on Electronic Commerce (EC 2001)*, Florida, USA, October 2001.
- [50] Site green grid, html, <http://www.thegreengrid.org>.
- [51] L. Hablot and P. Vicat-Blanc Primet O. Gluck, J.C. Mignot. Etude d'implémentations mpi pour une grille de calcul. In *RenPar'18/SympA'2008/CFSE'6*, Fribourg 2008.
- [52] Helmut Hlavacs, Georges Da Costa, and Jean-Marc Pierson. Energy consumption of residential and professional switches. *Computational Science and Engineering, IEEE International Conference on*, 1 :240–246, 2009.

- [53] Foster I. What is the grid? a three point checklist. GRIDToday, 2002.
- [54] Foster I. and Kesselman C. *The Grid : Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, inc., 1999.
- [55] Foster I., Kesselman C., Nick J., and Tuecke S. The physiology of the grid : An open grid services architecture for distributed systems integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, 2002.
- [56] Site web de ibm cluster systems management, <http://www-03.ibm.com/servers/eserver/clusters/software/csm.html>.
- [57] E. Jeannot. *Algorithmes et protocoles pour la gestion des données et des calculs dans les environnements distribués et hétérogènes*. PhD thesis, Habilitation à Diriger des Recherches de l'université Henri Poincaré, 2007.
- [58] Site web de la communauté jini, <http://www.jini.org>.
- [59] "Krishna Kant". *"Introduction to Computer System Performance Evaluation"*. "McGraw-Hill, inc", 1992.
- [60] J. O. Kephart and D. M. Chess. *The vision of autonomic computing*. Computer, 2003.
- [61] L. Kleinrock. *Queueing Systems. Vol. 1 : Theory*. Wiley, 1975.
- [62] P. Lorenz, J. Vagner Vital, B. Biscontini, and P. Russer. A grid-enabled time-domain transmission-line-matrix system for the analysis of complex electromagnetic structures. *IEEE Transactions on Microwave Theory and Techniques*, vol. 53, no.11, 3631–3637 (2005).
- [63] Chetty M. and Buyya R. Weaving computational grids : How analogous are they with electrical grids? *Computing in Science and Engineering*, 2002.
- [64] Quinson M. *Découverte automatique des caractéristiques et capacités d'une plate-forme de calcul distribué*. PhD thesis, ENS de Lyon, 2003.
- [65] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system : design, implementation, and experience. *Parallel Computing*, volume 30, issue 7, 2004.
- [66] B. Miegemolle. *Prédiction de comportement d'applications parallèles et placement à l'aide de modèles économiques sur une grille de calcul*. PhD thesis, Institut National des Sciences Appliquées de Toulouse, 2008.
- [67] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, 1996.
- [68] T. Monteil. *Etude de nouvelles approches pour les communications, l'observation et le placement de tâches dans l'environnement de programmation parallèle LANDA*. PhD thesis, ENSEEIHT, 1996.
- [69] T. Monteil and J.M. Garcia. Network-analyser : un système de collecte et de prédiction de l'état d'un réseau local pour le placement dynamique de processus. In *Journées de recherche, Université P.M'Curie Paris.*, pages pp87–90, 1995.
- [70] T. Monteil, J.M. Garcia, and P. Guyaux. Landa : une machine virtuelle parallèle. *Calculateurs Parallèles*, Vol 7 No 2, pages 119–137, 1995.
- [71] P. Morin. Coarse-grained parallel computing on heterogeneous systems. In *13th Annual ACM Symposium on Applied Computing (SAC'98), Atlanta (USA), 27 Février - 1er Mars 1998*.

- [72] A. W. Mu'alem and D.G. Feitelson. Utilization, predictability, workloads, and user run-time estimates in scheduling the ibm sp2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, Juin 2001.
- [73] définition du cloud computing sur wikipedia, [http://fr.wikipedia.org/wiki/cloud\\_computing](http://fr.wikipedia.org/wiki/cloud_computing).
- [74] 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*, 14(3) :228–251, 2000.
- [75] Site web d'opnet, <http://www.opnet.com>.
- [76] Patricia Pascal. Gestion des ressources sur des grappes d'ordinateurs avec qualité de service garantie. In *4ème Congrès Ecole Doctorale Système. Mai 2003, Toulouse*.
- [77] Patricia Pascal. *Gestion de ressources pour des services déportés sur des grappes d'ordinateurs avec qualité de service garantie*. PhD thesis, Université Paul Sabatier Toulouse, 2003.
- [78] K.R. Pattipati and J.L. Wolf. A file assignment problem model for extended local area network environments. In *Proceedings 10th international conference on distributed computing systems*, pp 554-561, 1990.
- [79] Site web de rocks cluster distribution, <http://www.rocksclusters.org>.
- [80] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. Softw. Eng.*, 21(8) :689–700, 1995.
- [81] L. J. Senger, M. J. Santana, and R. H. Carlucci Santana. An instance-based learning approach for predicting execution times of parallel applications. *Third International Information and Telecommunication Technologies Symposium, 2004*.
- [82] R. Sharrock, T. Monteil, P. Stolf, D. Hagimont, and L. Broto. Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS) vol.2 no.2, 2010*.
- [83] W. Smith, V. Taylor, and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. *Job Scheduling Strategies for Parallel Processing, Springer Verlag, 1999*.
- [84] Standard performance evaluation corporation, <http://www.spec.org>.
- [85] Daniel P. Spooner, Junwei Cao, Stephen A. Jarvis, Ligang He, and Graham R. Nudd. Performance-aware workflow management for grid computing. *Computer Journal*, 48(3) :347–357, 2005.
- [86] Jeffrey M. Squyres and Andrew Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag.
- [87] S. Richard, B. Miegemolle, T. Monteil, and J.M. Garcia. Performance of mpi parallel applications. In *Proceedings International Conference on Software Engineering Advances (ICSEA'2006), Papeete (France), 2006*.
- [88] Site web de sun cluster, <http://www.sun.com/software/cluster/index.xml>.
- [89] E. G. Talbi. Allocation dynamique de processus dans les systèmes distribués et parallèles : état de l'art. In *Université des Sciences et technologies de Lille Rapport 162, janvier 1995*.

- [90] Berstis V. Fundamentals of grid computing. Redpaper, IBM, 2002.
- [91] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, Vol. 6, 1997.
- [92] 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 (USA), Avril 2001.
- [93] Rich Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings 6th IEEE Symposium on High Performance Distributed Computing*, Portland Oregon., 1997.
- [94] L. Tianruo Yang and M. Guo. *High-performance computing : paradigm and infrastructure*. Wiley.
- [95] Yatpac homepage, [http ://www.yatpac.org/index.php](http://www.yatpac.org/index.php).