MINISTERE DE LA JEUNESSE, DE L'EDUCATION NATIONALE ET DE LA RECHERCHE

******

Institut National des Sciences Appliquees

TOULOUSE

Département de Génie Electrique & Informatique

******

DEA SYSTEMES INFORMATIQUES

ETUDE DES MODELES ECONOMIQUES POUR LES GRILLES DE CALCUL

LAAS - CNRS
Toulouse

MIEGEMOLLE Bernard

Génie Informatique et Industriel
Orientation Génie Informatique

Septembre 2004

Avant propos

Je remercie Monsieur Malik GHALLAB pour son accueil dans le Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) dont il assure la direction.

Je remercie Monsieur Gérard AUTHIE, responsable du groupe Réseaux et Systèmes de Télécommunications (RST), ainsi que Monsieur Jean-Marie GARCIA, responsable de l'équipe ACE, pour m'avoir permis d'effectuer ce stage.

J'exprime toute ma reconnaissance à Monsieur Thierry MONTEIL pour m'avoir fait confiance. Grâce à ses compétences et à ses grandes qualités humaines, j'ai pu bénéficier du meilleur encadrement possible. Qu'il trouve ici le témoignage de ma plus grande considération.

Je tiens également à remercier Monsieur Samuel RICHARD, doctorant au LAAS, de m'avoir aidé et conseillé tout au long de mon travail. Ses grandes compétences techniques m'ont souvent été très précieuses pour mener à bien la tâche qui m'a été confiée.

Merci enfin à tous les membres du groupe RST ainsi qu'à l'ensemble du personnel du LAAS pour leur accueil chaleureux.

Table des matières

Introduction **4**

1 Le grid computing **5**
1.1 Objectifs du grid computing . . . 6
1.2 Caractéristiques d'une grille . . . 7
1.3 Architecture d'une grille . . . 8
1.4 Domaines d'application . . . 9
1.5 Exemples de grilles . . . 9
1.6 Conclusion . . . 10

2 Globus **11**
2.1 Présentation générale . . . 11
2.2 Les différents composants de la Globus Toolkit . . . 12
2.2.1 Le gestionnaire de ressources . . . 12
2.2.2 Le service d'information . . . 14
2.2.3 Le gestionnaire de données . . . 15
2.2.4 Le service de sécurité . . . 16
2.3 OGSA et les Grid Services . . . 17
2.3.1 La norme OGSA . . . 17
2.3.2 Les Grid Services . . . 18
2.4 Conclusion . . . 19

3 Modèles économiques pour la gestion des ressources d'une grille de calcul **20**
3.1 Introduction . . . 20
3.2 Fonctionnement général . . . 21
3.3 Les différents modèles économiques existants . . . 22
3.3.1 Modèle de marché . . . 22
3.3.2 Modèle des négociations . . . 22
3.3.3 Modèle de l'appel d'offres . . . 23
3.3.4 Modèle de la vente aux enchères . . . 24
3.3.5 Modèle du partage proportionnel des ressources . . . 25
3.4 Etablissement des prix et mécanismes de paiement . . . 25
3.4.1 Ressources à facturer . . . 25
3.4.2 Etablissement du prix d'une ressource . . . 25
3.4.3 Mécanismes de paiement . . . 26
3.5 Conclusion . . . 26

4 Mise en œuvre d'un modèle économique simple **27**
4.1 Présentation du contexte . . . 27
4.1.1 Le projet CASP et Aroma . . . 27
4.1.2 Objectifs fixés . . . 28
4.2 Etude théorique . . . 29

4.2.1 Caractéristiques du modèle économique 29
4.2.2 Définition des variables 29
4.2.3 Modèle mathématique 32
4.2.4 Résultats 34
4.2.5 Conclusion 40
4.3 Mise en œuvre 41
4.3.1 Sauvegarde des paramètres de coût dans une base de données 41
4.3.2 Prise en compte des mécanismes de facturation au niveau de l'ordonnanceur 43
4.3.3 Conclusion 46

Conclusion **47**

Table des figures

1.1 Répartition des ressources d'une grille entre différents domaines . . . 7
1.2 Architecture en couches d'une grille . . . 8

2.1 Les trois piliers de la Globus Toolkit . . . 12
2.2 Le GRAM au cœur d'une architecture en sablier . . . 13
2.3 Architecture du service d'information . . . 14
2.4 Modes de transfert du protocole GridFTP . . . 15
2.5 Principe de l'authentification . . . 17
2.6 Principe de l'invocation d'un Web Service . . . 19

3.1 Principe du modèle de marché . . . 23
3.2 Principe du modèle des négociations . . . 23
3.3 Principe du modèle de l'appel d'offres . . . 24
3.4 Principe du modèle de la vente aux enchères . . . 24

4.1 Communications entre les fournisseurs de services et les clients . . . 28
4.2 Minimisation de la durée d'exécution de l'application . . . 34
4.3 Minimisation de la durée d'exécution de l'application avec prise en compte de la charge des processeurs . . . 35
4.4 Minimisation du coût de l'application (influence de la puissance des processeurs) . . . 36
4.5 Minimisation du coût de l'application (utilisation de coefficients de coût locaux) . . . 37
4.6 Minimisation du coût de l'application (influence du temps processeur consommé) . . . 38
4.7 Minimisation d'un compromis entre coût et durée d'exécution . . . 39
4.8 Minimisation d'un compromis entre coût et durée d'exécution (cas complet) . . . 40
4.9 Fenêtre permettant de fixer les paramètres de coût communs aux machines . . . 41
4.10 Principe de la sauvegarde des paramètres de coût dans la base de données . . . 42
4.11 Fenêtre permettant d'ajuster le coût de chacune des machines . . . 42
4.12 Communications échangées lors du placement d'une application . . . 43
4.13 Interface permettant à l'utilisateur de renseigner les paramètres décrivant l'application . . . 44
4.14 Interface permettant à l'utilisateur de suivre l'évolution de l'application . . . 45

Introduction

Le stage que j'ai effectué dans le cadre du DEA fait suite au projet de fin d'études que j'ai pu mener au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS). Ce projet, intitulé « Services Dynamiques pour l'Utilisation de Grilles de Machines et de Réseaux », m'a permis de découvrir le domaine du calcul sur grilles, en travaillant sur un outil en cours de développement au LAAS : Aroma.

Le stage de DEA se situe également dans le domaine du *grid computing*, même si ses objectifs sont totalement différents de ceux du projet de fin d'études. L'objectif principal de cette partie du stage a, en effet, été de réaliser une étude des différents modèles économiques permettant la gestion des ressources d'une grille, afin d'aboutir à la mise en place d'un modèle pour Aroma.

Le présent document est découpé en quatre parties. Dans un premier temps, nous présenterons ce qu'est le *grid computing*, quels sont ses objectifs, et nous montrerons en quoi consiste une grille de calcul. Ensuite, nous décrirons les principaux constituants de l'infrastructure logicielle permettant le bon fonctionnement d'une grille. Nous réaliserons cette étude au travers d'un exemple : le projet américain Globus. Dans une troisième partie, nous montrerons que les modèles économiques du marché réel peuvent servir pour la gestion des ressources d'une grille de calcul. Enfin, la dernière partie nous permettra d'approfondir l'étude des modèles économiques en mettant en place notre propre modèle.

Chapitre 1

Le grid computing

L'augmentation constante des besoins en termes de puissance de calcul informatique a toujours été un défi auquel la communauté informatique s'est confrontée. Le calcul scientifique ou financier, la modélisation ou bien la réalité virtuelle sont autant d'exemples pour lesquels il est nécessaire de disposer d'une grande puissance de calcul.

Même si les évolutions technologiques de ces dernières années ont permis d'aboutir à la création de machines de plus en plus puissantes, celles-ci ne fournissent toujours pas suffisamment de puissance pour effectuer des calculs d'une complexité trop élevée, ou traitant un trop grand nombre de données.

Le calcul parallèle, ou distribué, est une réponse à ce problème [Quinson, 2003]. Il s'agit de répartir le calcul sur un ensemble de machines, reliées entre elles par des réseaux rapides, afin d'augmenter les performances pour l'effectuer. Cette idée est à l'origine du calcul sur grille, ou *grid computing*.

Une grille de calcul est une infrastructure matérielle et logicielle qui fournit un accès consistant et peu onéreux à des ressources informatiques [Foster & Kesselman, 1999]. Le but est ainsi de fédérer des ressources provenant de diverses organisations désirant collaborer en vue de faire bénéficier les 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 de calcul [Foster, 2002]. 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).

Une analogie forte existe entre le développement des grilles de calcul et celui des grilles d'électricité (*power grids*) au début du XXème siècle [Chetty & Buyya, 2002]. 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 [Foster & Kesselman, 1999]. Les composants formant le réseau électrique sont hétérogènes, et la complexité induite est totalement masquée à l'utilisateur final. Ainsi, une grille de calcul possède les mêmes propriétés d'hétérogénéité des ressources la constituant, et de transparence vis-à-vis de l'utilisateur final.

Quels sont les facteurs qui ont permis l'émergence de cette nouvelle technologie ? Depuis quelques années, nous assistons à une explosion de la puissance des stations de travail pour des coûts moindres pour le grand public. Des réseaux à haut débit sur longues distances ont également vu le jour, avec notamment l'Internet grand public. Ces deux facteurs ont été essentiels pour le développement des grilles de calcul, une grille étant constituée d'un ensemble de machines interconnectées par des réseaux classiques, notamment l'Internet. Cette solution à bas prix permet d'introduire un certain degré de transparence dans l'usage des ressources de calcul et de stockage.

1.1 Objectifs du grid computing

Le calcul sur grille possède plusieurs objectifs [Berstis, 2002] :

- **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. Bien entendu, lancer une application sur un ordinateur inactif suppose que celui-ci possède les équipements matériels et logiciels nécessaires au bon fonctionnement de l'application. Il est à noter que les ressources que l'on traite ici peuvent aussi bien être des cycles de processeur que des capacités de stockage (mémoires vives ou mémoires permanentes).

- **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. Cependant, toutes les applications ne pourront pas s'exécuter en parallèle. C'est le cas lorsque les tâches qui la composent sont trop dépendantes les unes des autres.

- **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 ressources, tels que des équipements spéciaux, des logiciels, et bien d'autres services. Certaines machines peuvent, par exemple, posséder des logiciels à licence très élevée requis par une application. Ainsi, exécuter l'application sur ces machines permet de bénéficier de tels logiciels. Si une machine est reliée à des équipements spécifiques, elle pourra être utilisée par les autres machines de la grille pour permettre le partage de ces équipements.

- **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 relativement inactives.

- **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 à un endroit 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 retournent pour plus de sûreté.

1.2 Caractéristiques d'une grille

Le *grid computing* consiste donc à fédérer des ressources de calcul et de stockage géographiquement réparties, en vue de permettre leur utilisation de manière transparente pour tout client de la grille. Nous allons maintenant examiner les caractéristiques principales d'une grille de calcul, et évaluer les difficultés que pose leur gestion [Quinson, 2003].

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. Il est donc indispensable de gérer au mieux cet aspect en fournissant des interfaces standards pour qu'elles puissent être utilisées de façon transparente.

Les ressources sont géographiquement distribuées et sont possédées par différentes organisations indépendantes. Elles appartiendront donc à plusieurs domaines administratifs distincts, ayant chacun sa propre politique de gestion et de sécurité (figure 1.1). Ainsi, les personnes chargées d'administrer la grille n'auront pas forcément de privilèges particuliers sur les machines des différents domaines administratifs. Il est donc indispensable de mettre au point des méthodes d'administration particulières ne nécessitant aucun privilège sur les machines cibles.

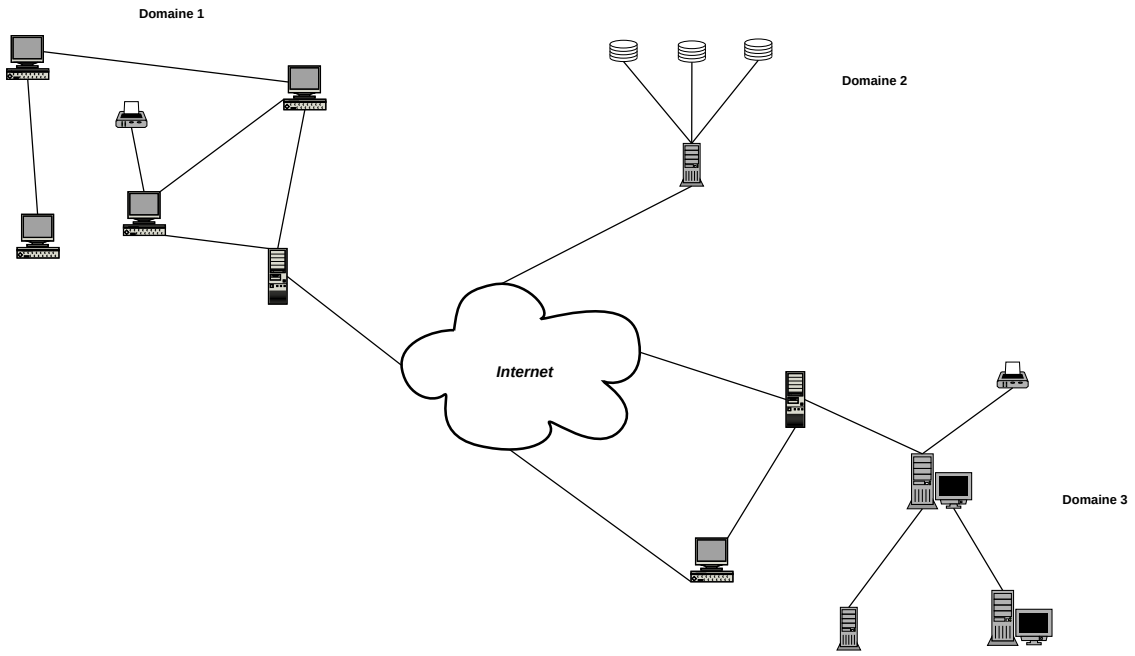


FIG. 1.1: Répartition des ressources d'une grille entre différents domaines

La capacité d'extensibilité ou de passage 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 d'une dizaine de millions de ressources. Ce problème de dimensionnement pose de nouvelles contraintes sur les applications et les algorithmes de gestion des ressources.

Enfin, il est indispensable de ne pas oublier la nature dynamique des ressources d'une grille. Un changement de la constitution de la grille peut survenir très fréquemment. Ceci suppose que les applications peuvent facilement s'adapter aux changements du nombre de ressources, et que des mécanismes de tolérance aux fautes soient mis en place.

1.3 Architecture d'une grille

L'architecture d'une grille peut être vue de plusieurs façons. Nous choisirons ici de la représenter selon une architecture en couches [Chetty & Buyya, 2002] (figure 1.2).

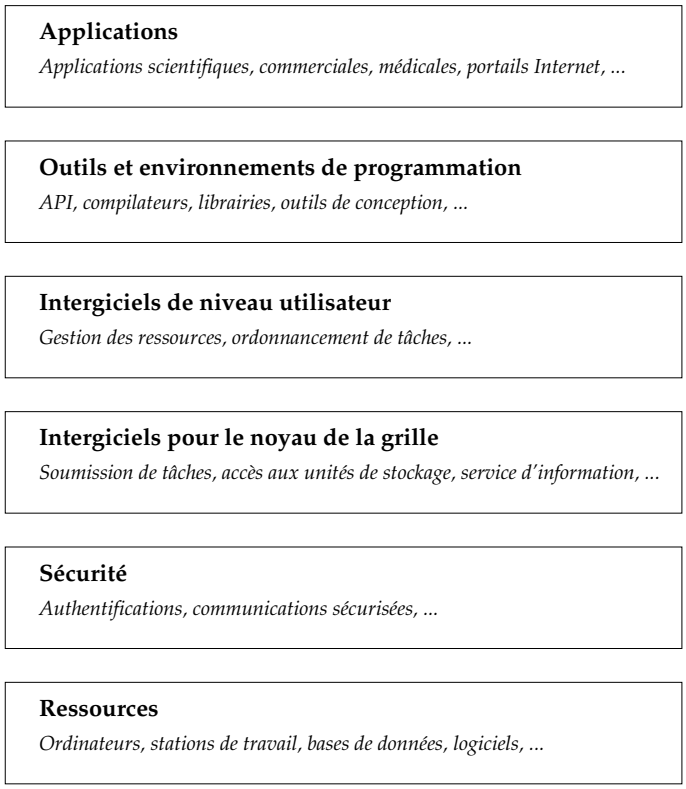


FIG. 1.2: Architecture en couches d'une grille

La couche de plus bas niveau correspond aux ressources elles-mêmes, gérées par des gestionnaires locaux, et interconnectées au travers de réseaux locaux et grande distance. Cette couche contient ainsi toute l'infrastructure matérielle de la grille (ordinateurs personnels, stations de travail, unités de stockage, bases de données, etc.), mais aussi les systèmes de gestion de ressources (tels que Condor, LSF ou bien PBS).

La seconde couche fournit tous les mécanismes nécessaires à la sécurité de la grille. Des procédures d'identification et d'authentification sont ainsi mises en place afin de pouvoir accéder aux ressources constituant la grille. Le degré de sécurisation d'une ressource peut varier selon son importance ou sa criticité. Ainsi, il peut exister des ressources pour lesquelles aucune authentification n'est nécessaire pour les utiliser.

La troisième couche fournit les intergiciels[1] nécessaires au noyau de la grille. Elle offre des outils permettant la soumission de tâches, l'accès aux unités de stockage et des services d'information répertoriant dynamiquement les ressources disponibles et leur état.

La quatrième couche regroupe des intergiciels de niveau utilisateur. Elle concerne les outils de gestion des ressources et les services d'ordonnancement. Ces ordonnanceurs auront pour rôle de choisir la machine la plus appropriée pour placer les tâches qui lui sont confiées, compte tenu de l'état de charge des différentes ressources de la grille.

Des outils et environnements de programmation pour le développement d'applications destinées aux grilles de calcul sont fournis par la cinquième couche. On y trouve ainsi des interfaces de programmation (API), des compilateurs, des librairies ou bien encore des outils de conception d'applications parallèles adaptées aux grilles.

La sixième et dernière couche regroupe enfin les applications elles-mêmes. Il peut s'agir d'applications scientifiques tout comme d'applications commerciales, médicales ou bien des portails Internet.

1.4 Domaines d'application

Il existe cinq grands types de domaines d'application pour lesquels les grilles peuvent être utilisées [Foster & Kesselman, 1999] :

- **Le calcul intensif distribué :** Il s'agit d'utiliser les grilles de calcul en vue d'agréger une importante quantité de ressources nécessaires à certaines applications. Une telle puissance de calcul ne peut être obtenue qu'en additionnant les capacités de plusieurs machines. Des simulations interactives dans le domaine militaire, ou bien des simulations de processus physiques complexes sont des exemples d'application du calcul intensif distribué.
- **Le calcul à haut débit :** Dans ce contexte, la grille permet d'ordonnancer en parallèle un grand nombre de tâches peu couplées, voire totalement indépendantes, dans le but d'utiliser les cycles processeurs inutilisés des machines inactives. Parmi les différentes applications, nous pouvons citer toutes les résolutions de problèmes cryptographiques et l'analyse du génome.
- **Le calcul à la demande :** Ce domaine concerne les applications pour lesquelles les grilles de calcul sont un moyen de disposer temporairement de ressources dont l'utilisation permanente ne serait pas rentable. En outre, parmi ces applications figurent les applications scientifiques nécessitant l'utilisation de matériels spécifiques onéreux.
- **Le traitement intensif de données :** Le rôle des grilles de calcul est, dans ce cas, de produire de nouvelles informations à partir de données géographiquement distribuées. La grille sera alors en charge de stocker la masse d'information ainsi générée. Des exemples d'applications sont la production d'une carte de l'univers, ou bien encore les prévisions météorologiques.
- **Le calcul collaboratif :** Les applications collaboratives ont pour objectif de permettre les interactions entre humains afin d'autoriser l'utilisation partagée de ressources telles que des bases de données ou bien des simulations.

1.5 Exemples de grilles

Les grilles de calcul, même si elles constituent une nouvelle technologie en cours de développement, ont plusieurs exemples concrets d'utilisation à leur actif [Ferreira & al.].

[1] Un intergiciel (*middleware*) est une classe de logiciels qui assure l'intermédiaire entre les applications et les systèmes d'exploitation présents sur les machines de la grille.

Le premier exemple de grille est celle constituée dans le domaine de l'imagerie numérique médicale, notamment utilisée dans le domaine de la cancérologie. Grâce à cette grille, quatre hôpitaux se partagent des images numériques de cancers, offrant, d'une part, une aide aux diagnostics, et d'autre part, des outils pour faire avancer la recherche dans ce domaine.

Un second exemple de grille est la *ZetaGrid*. Elle est constituée de plus de trois mille machines volontaires reliées par Internet en vue d'essayer de vérifier l'hypothèse formulée en 1859 par Riemann, et qui reste à ce jour l'un des plus importants problèmes mathématiques. Ainsi, pour participer à cette grille, il suffit de télécharger librement un client, dont le rôle est d'exécuter des tâches sur la machine dès que l'économiseur d'écran de celle-ci devient actif.

Il existe encore de nombreux projets de grilles, par exemple dans le domaine financier, ou bien encore pour réaliser des simulations dans l'industrie aérospatiale, mais nous terminerons cette section en abordant le projet connu sous le nom de *SETI@home*. Ce projet a pour but de rechercher une éventuelle trace d'intelligence extraterrestre à partir d'observations réalisées par un radio-télescope. Une grille, constituée de cinq cent mille ordinateurs, a ainsi été mise en place. Il s'agit à ce jour de la plus vaste grille jamais déployée au monde.

1.6 Conclusion

Nous venons de voir qu'une grille est un ensemble de machines reliées par des réseaux, dont le but est de mettre en commun leurs ressources afin d'en optimiser l'utilisation. Ces infrastructures peuvent permettre d'utiliser les périodes d'inactivité des ordinateurs pour leur confier des tâches à exécuter, de lancer des applications en parallèle, ou bien encore de bénéficier d'une capacité de stockage qu'un seul ordinateur ne peut offrir.

Cependant, il convient de rester prudent avec cette technologie. Une grille n'est en aucun cas un environnement qui permet d'exécuter n'importe quelle application mille fois plus vite, sans avoir à payer aucune machine ou logiciel supplémentaire [Berstis, 2002]. En effet, il faut admettre que toutes les applications ne sont pas adaptées aux grilles de calcul, puisque certaines d'entre elles ne peuvent tout simplement pas être exécutées en parallèle.

Après avoir défini ce qu'est une grille de calcul, nous pouvons maintenant examiner de plus près l'infrastructure logicielle nécessaire à son bon fonctionnement, c'est-à-dire les intergiciels. Nous réaliserons cette étude en nous concentrant sur un des intergiciels existants : Globus.

Chapitre 2

Globus

Une grille de calcul a pour ambition d'offrir aux différents utilisateurs un accès transparent aux différentes ressources qui la composent. Ceci ne peut être possible que s'il existe une infrastructure logicielle adressant les nombreux problèmes qui se posent dans un tel contexte (sécurité, gestion des ressources, etc.).

Dans la section 1.3, nous avons vu que c'est le rôle des couches correspondant aux intergiciels. Il existe actuellement de nombreux intergiciels, souvent développés dans le cadre de projets de recherche (Globus, Legion, Unicore, Sun Grid Engine, etc.).

Nous étudierons, dans ce chapitre, le fonctionnement des intergiciels en utilisant l'exemple de Globus. Après une description générale de cet outil, nous examinerons ses principaux composants. Puis nous étudierons la norme OGSA, mise en œuvre dans la troisième version de la *Globus Toolkit*, ainsi que la notion de *Grid Services* qu'elle a introduite.

2.1 Présentation générale

La *Globus Toolkit* est un ensemble d'outils et de logiciels *open-sources* permettant de concevoir et de mettre en œuvre des grilles de calcul et les applications qui leur sont destinées. Actuellement dans sa version 3, la *Globus Toolkit* a été mise au point par la *Globus Alliance*, un groupe de recherche, dirigé par Ian Foster. Ce groupe de recherche est basé à l'*Argonne National Laboratory*, aux Etats-Unis, et est spécialisé dans les technologies de *grid computing*.

L'objectif principal de la *Globus Toolkit* est de fournir aux différents utilisateurs d'une grille une API leur permettant d'accéder de façon transparente aux ressources qui leur sont offertes. Les outils ainsi mis à disposition des utilisateurs ont pour but d'adresser plusieurs problèmes auxquels est souvent confronté le *grid computing*. Parmi ceux-ci figurent [Foster & Kesselman, 1997] :

- la localisation et l'allocation de ressources,
- les communications,
- la découverte d'informations sur les ressources,
- la sécurité,
- la gestion et l'accès aux données.

De manière générale, la composition de la *Globus Toolkit* peut être représentée par les trois piliers de la figure 2.1.

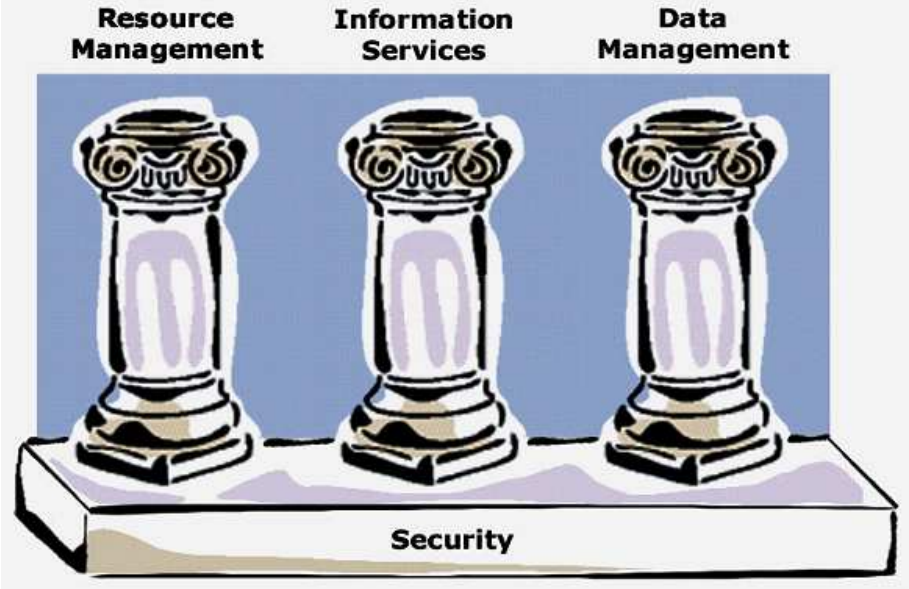


FIG. 2.1: Les trois piliers de la Globus Toolkit

Ainsi, trois composants essentiels de la *Globus Toolkit* sont à distinguer :

- Le « Resource Management » fait allusion au service d'allocation de ressources. Il est principalement composé du GRAM (*Grid Resource Allocation Manager*) et du GASS (*Globus Access to Secondary Storage*).
- Le pilier « Information Services » fait référence au service d'information de la *Globus Toolkit*, c'est-à-dire au MDS-2 (*Meta Directory Service*) dont le rôle est de collecter les informations concernant l'état de la grille au sein d'une base de données.
- Le service de gestion des données présentes sur une grille est représenté par le pilier « Data Management ». Des outils tels que GridFTP ou bien RFT (*Reliable File Transfer*) offrent aux utilisateurs la possibilité d'accéder à ces données et de les modifier.

Les trois piliers de la *Globus Toolkit* reposent sur un service de sécurité, nécessaire à la viabilité de la grille et des ressources qu'elle héberge. Cette sécurité est assurée par le GSI (*Grid Security Infrastructure*) qui offre les mécanismes nécessaires à la réalisation d'authentifications et de communications sécurisées au travers d'un réseau étendu.

2.2 Les différents composants de la Globus Toolkit

Dans cette section, nous allons étudier les principaux composants de la *Globus Toolkit*, correspondant aux trois piliers de la figure 2.1. Nous examinerons tout d'abord le fonctionnement du gestionnaire de ressources. Ensuite, nous nous intéresserons au service d'information, puis au gestionnaire de données. Nous étudierons enfin le fonctionnement général du service de sécurité.

2.2.1 Le gestionnaire de ressources

La gestion des ressources au sein de la Globus Toolkit est assurée par deux entités : le GRAM, chargé de l'allocation des ressources, et le GASS, dont le rôle est de gérer le transfert des fichiers utilisés par une application.

Le GRAM

Le GRAM (*Grid Resource Allocation Manager*) est le service de gestion de l'allocation des ressources de la *Globus Toolkit*. Il s'agit d'une API dont le rôle est de mettre à disposition des utilisateurs les outils nécessaires à l'exécution d'applications à distance, et de gérer les ressources qui leur sont associées.

La figure 2.2 montre la place du GRAM au sein de l'architecture en couches de Globus [Foster & Kesselman, 1998]. Au niveau du GRAM, un rétrécissement se forme (on parle alors d'architecture en sablier). Ceci traduit le fait que le GRAM possède une API simple et bien définie, fournissant un accès uniforme aux diverses implémentations des services de bas niveau. Des services de haut niveau peuvent alors être définis en s'appuyant sur cette interface, sans se préoccuper de la constitution de la couche de bas niveau.

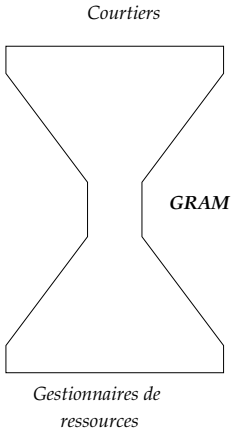


FIG. 2.2: Le GRAM au cœur d'une architecture en sablier

Sur une grille, plusieurs GRAM sont généralement déployés, chacun étant responsable d'un ensemble de ressources soumises à une même politique de gestion qui est spécifique au site dans lequel elles se trouvent. Chaque GRAM s'appuie sur un gestionnaire de ressources local, tel que Condor, LSF (*Load Sharing Facility*), ou bien PBS (*Portable Batch System*), chargé d'appliquer cette politique. Ainsi, pour chaque site, les administrateurs peuvent bénéficier de l'API du GRAM tout en étant libres de choisir le gestionnaire de ressources qu'ils veulent. De ce fait, aucun gestionnaire local de ressources n'est fournit avec la *Globus Toolkit*.

Lorsque le GRAM devra allouer des ressources à une application, il fera appel au gestionnaire local qui se chargera d'ordonnancer les différentes tâches qui la composent sur les ressources dont il est responsable. Notons que des courtiers (*Resource Brokers*) peuvent être placés au dessus des différents GRAM afin d'appliquer des politiques globales de gestion de ressources. Lorsqu'une application devra être exécutée, le courtier identifiera un ensemble de ressources, et donc un ensemble de GRAM, susceptibles de satisfaire la demande.

Le GASS

Le GASS (*Globus Access to Secondary Storage*) entre lui aussi en jeu lors de l'exécution d'une application sur la grille. Son rôle est de transférer les fichiers nécessaires à l'exécution d'une tâche de l'application, d'une machine distante vers la machine sur laquelle la tâche est placée [Lock, 2002].

Comment cela fonctionne-t-il ? Toutes les ressources additionnelles requises par une tâche sont répertoriées dans la requête de placement de la tâche. En recevant cette requête, le GRAM détermine si ces ressources manquent sur la machine affectée à la tâche. Si c'est la cas, un serveur GASS est créé sur cette machine dans le but de les récupérer. Le serveur GASS autorise également les utilisateurs à placer des fichiers eux-mêmes dans un cache mis en place par celui-ci. Cette opération nécessite des mécanismes de sécurité permettant l'authentification des utilisateurs.

2.2.2 Le service d'information

Le service d'information de la *Globus Toolkit* est mieux connu sous le nom de *Meta Directory Service* (MDS). Son rôle est de collecter les informations concernant l'état de la grille au sein d'une base de données, et de les mettre à disposition des utilisateurs sur demande.

Ces informations peuvent être de plusieurs natures :

- configuration des ressources, c'est-à-dire les informations statiques les concernant (par exemple : fréquence du processeur, nombre de processeurs, quantité de mémoire, etc.),
- état instantané des ressources, c'est-à-dire les informations dynamiques les concernant (par exemple : charge du processeur, nombre de processeurs utilisés, quantité de mémoire utilisée, etc.),
- informations sur les applications (par exemple : besoins en termes de processeur, de mémoire, etc.).

Ainsi, le MDS permet de gérer l'aspect dynamique d'une grille de calcul, en permettant aux composants de la *Globus Toolkit*, aux outils de programmation, et aux applications d'être capables d'adapter leur comportement aux changements de la structure ou de l'état du système [Foster & Kesselman, 1998].

Ce service est essentiellement composé de deux entités : le GRIS qui répertorie les informations sur les machines qui y sont enregistrées, et le GIIS chargé d'indexer les serveurs GRIS [Lock, 2002] (figure 2.3). Nous allons décrire successivement ces deux composants.

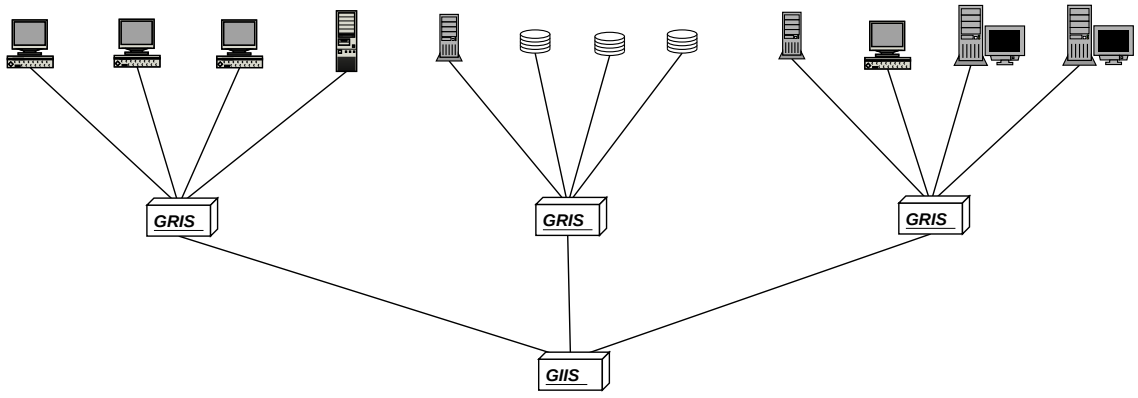


FIG. 2.3: Architecture du service d'information

Le GRIS

Le GRIS (*Grid Resource Information Service*) permet la sauvegarde d'informations, statiques ou dynamiques, provenant des ressources qui y sont enregistrées. Un seul et même serveur GRIS ne contient jamais les informations concernant toutes les machines de la grille. Ceci permet d'une part une meilleure tolérance aux fautes, et d'autre part de moins surcharger le GRIS (et donc d'avoir des temps de réponse moins élevés). Ainsi, il existe plusieurs serveurs GRIS sur une même grille.

Dès lors, un utilisateur quelconque, recherchant des informations sur une machine particulière, devra forcément savoir à quel GRIS s'adresser pour les récupérer. Globus fournit pour cela un second outil, le GIIS.

Le GIIS

Le GIIS (*Grid Index Information Service*) permet d'indexer les serveurs GRIS d'une grille. Chaque GRIS s'enregistre, dès son démarrage, auprès d'un serveur GIIS. Celui-ci contient des informations concernant la localisation des GRIS dans la grille, ainsi que les noms des machines enregistrées auprès de chaque GRIS. Il peut également contenir des informations normalement enregistrées au niveau des GRIS. Le GIIS fournit ainsi une image cohérente de la globalité de la grille.

Il est à noter que la présence d'un seul serveur GIIS sur une grille constituerait un point fragile. A ce propos, des serveurs secondaires sont mis en place afin d'assurer une meilleure tolérance aux fautes.

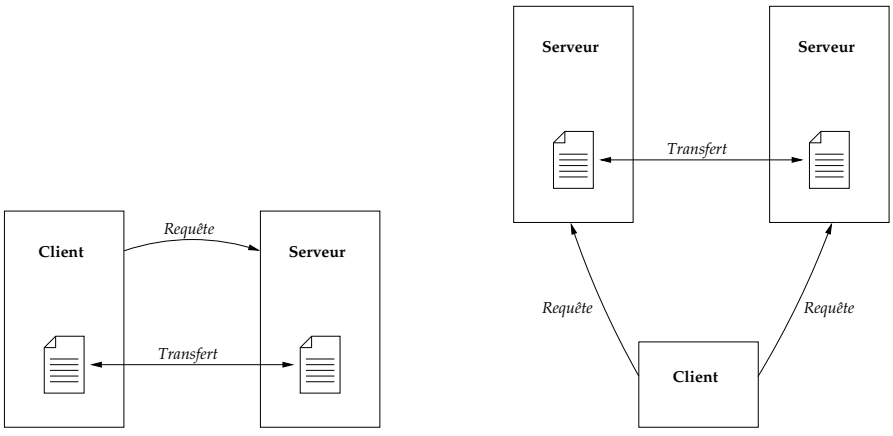
2.2.3 Le gestionnaire de données

Le service de gestion des données est principalement en charge de leurs transferts au sein de la grille. C'est dans ce but que le GridFTP a été mis en place [Ferreira & al.]. Il est à noter que le terme de « GridFTP » désigne à la fois le protocole, le serveur ainsi que l'ensemble des outils permettant d'effectuer des transferts de données fiables entre les différentes machines de la grille.

Le protocole GridFTP est une extension du protocole FTP standard, qui permet de s'adapter aux grilles de calcul, et notamment aux mécanismes de sécurité requis.

Il existe deux types de transfert (figure 2.4) :

- le transfert de fichiers standard : des fichiers sont transférés entre le client et le serveur GridFTP.
- le transfert impliquant une troisième entité : le client peut demander qu'un transfert de fichiers soit effectué entre deux serveurs de la grille.



(a) Mode FTP standard

(b) Mode FTP étendu

FIG. 2.4: Modes de transfert du protocole GridFTP

Il est à noter que le GASS (cf. section 2.2.1) est un sous-ensemble des outils GridFTP.

2.2.4 Le service de sécurité

Dans une grille de calcul, l'aspect lié à la sécurité du système est fondamental pour la viabilité de la grille à long terme. En effet, il est nécessaire de contrôler un tel système afin d'éviter toute intrusion pouvant mettre en péril l'intégrité des ressources de la grille.

Dans la *Globus Toolkit*, le service gérant la sécurité de la grille est le GSI (*Grid Service Infrastructure*). Comment fonctionne-t-il [Lock, 2002] ?

Cryptage des données

Afin d'éviter que n'importe qui puisse accéder aux données transitant entre les différentes machines de la grille, un mécanisme de cryptage des données a été mis en place. Il repose sur le cryptage à clé publique. Chaque entité de la grille possède deux clés : une clé publique qui permettra à toute autre entité de crypter les données qui lui sont destinées, et une clé privée, qu'elle seule possède, lui permettant de décrypter ces données.

Ainsi, grâce à ce mécanisme, il est garanti que seul le destinataire d'un message peut lire ce message. Toute personne désirant détourner des données ne pourra pas les décrypter, puisqu'elle n'est pas sensée détenir la clé privée du destinataire.

Le cryptage des données peut également être effectué de manière inversée : le message est codé à l'aide d'une clé privée. Dans ce cas, il ne peut être décodé que par la clé publique correspondante. Ceci permet de s'assurer non pas de l'identité du destinataire, mais au contraire de celle de l'émetteur.

Certificats X.509

Un certificat X.509 est un certificat numérique permettant d'identifier un utilisateur ou bien une machine de la grille. Pour une personne, il contient des informations telles que son nom, son identifiant ou bien l'organisation à laquelle elle appartient. Il contient également la clé publique qui lui est associée.

Il est évident que n'importe qui peut créer un certificat et prétendre appartenir à la grille. Pour éviter ceci, une autorité de certification doit examiner tout certificat nouvellement créé et le signer, afin qu'il soit valide.

Authentification et autorisation

Grâce aux mécanismes de cryptage et aux certificats X.509, il est possible d'authentifier formellement toute entité de la grille. Le certificat, s'il est signé, nous permet d'être sûr que l'entité se présentant sous ce certificat appartient à la grille. Avant de communiquer, un protocole basé sur le cryptage est mis en place afin de s'assurer que les entités communicantes sont bien qui elles prétendent. La figure 2.5 décrit cette phase d'authentification. Notons que sur ce schéma, l'hôte B a bien authentifié l'hôte A. Il faut ensuite refaire de même pour que l'hôte A puisse authentifier l'hôte B.

Ensuite, même si une personne peut utiliser la grille, il n'est pas dit qu'elle puisse utiliser la totalité des machines disponibles. Ainsi, lorsqu'un utilisateur quelconque désire se servir d'une machine, et après succès de la phase d'authentification, il est nécessaire de vérifier s'il est autorisé à utiliser la machine.

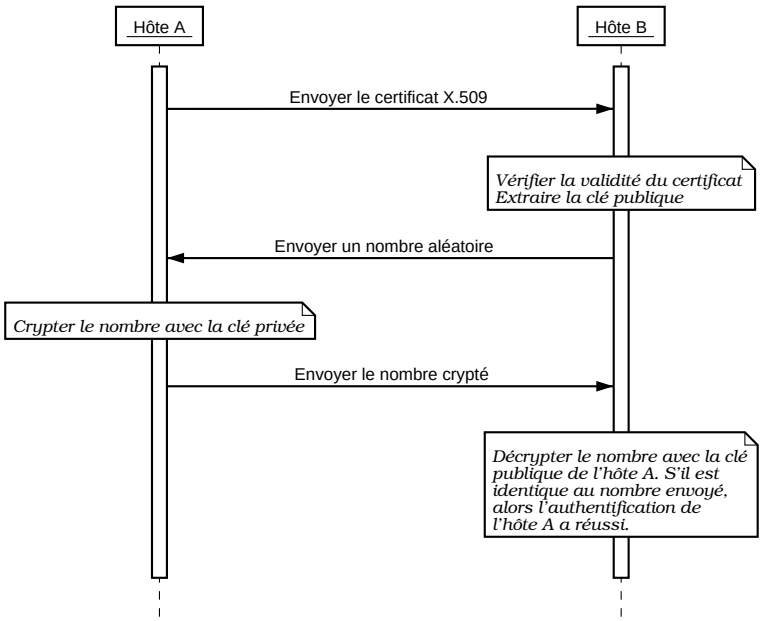


FIG. 2.5: Principe de l'authentification

Pour cela, chaque machine possède un fichier, appelé `grid-map`, qui contient la liste des utilisateurs pouvant l'utiliser (le certificat de chacun des utilisateurs possède une entrée dans ce fichier). Ainsi, l'utilisateur fournira son certificat à la machine qu'il désire utiliser. Un protocole d'authentification sera établi. Si l'opération est un succès, on vérifiera qu'une entrée correspondant au certificat fourni existe dans le fichier `grid-map`. Si c'est le cas, la requête de l'utilisateur peut être transmise à la machine.

2.3 OGSA et les Grid Services

2.3.1 La norme OGSA

OGSA (*Open-Grid Service Architecture*) est une norme mise en place dans la troisième version de la *Globus Toolkit*. Elle spécifie principalement [Foster & al, 2002] que toute entité d'une grille de calcul (ressources de calcul et de stockage, réseaux, programmes, bases de données, etc.) est vue comme un service de grille (*Grid Service*). Il s'agit ainsi d'offrir une vue suffisamment abstraite sur chacun des constituants de la grille pour que son utilisation puisse être réalisée de manière transparente.

L'enjeu principal de cette norme est ainsi de définir ce qu'est un *Grid Service*, quelles sont ses interfaces, et quel est son comportement. Elle peut donc être vue comme un modèle de service.

La norme OGSA se propose ainsi de :

- définir des mécanismes pour la création et la découverte d'instances temporaires de *Grid Services*,
- permettre un certain degré de transparence sur la localisation des instances de services,
- définir les mécanismes nécessaires à la création de systèmes distribués sophistiqués, et notamment les mécanismes de notification ainsi que la gestion du cycle de vie des services.

Pour résumer, OGSA définit comment un service est créé, comment il est nommé, comment sa durée de vie est déterminée, ou bien encore comment communiquer avec lui. Cependant, OGSA ne définit en aucun cas la nature du service rendu, ni comment ce service est réalisé.

2.3.2 Les Grid Services

Le terme de *Grid Service* désigne, de manière générale, un service disponible dans un environnement de grille [Sandholm & Gawor, 2003]. Dans le cadre de la *Globus Toolkit*, il désigne plus particulièrement les services qui respectent les spécifications imposées par la norme OGSI (*Open Grid Services Infrastructure*) [Tuecke & al, 2003], et qui s'exposent sur la grille grâce à une interface en WSDL.

Ces services, sur lesquels la version actuelle de la *Globus Toolkit* est basée, sont largement inspirés des *Web Services*[1]. Une grille étant un ensemble de machines interconnectées par des réseaux tels que l'Internet, cette technologie d'invocation de services à distance, offre deux avantages majeurs :

- Elle utilise des langages XML standards. Les services sont ainsi totalement indépendants de la plate-forme sur laquelle ils tournent ainsi que des langages utilisés pour écrire clients et serveurs.
- Les messages transmis par ces services utilisent le protocole HTTP, particulièrement adapté au monde de l'Internet.

Les *Web Services* reposent sur trois standards :

- SOAP (*Simple Object Access Protocol*) : fournit les moyens nécessaires à l'échange de messages entre un fournisseur de services et ses clients. SOAP définit des conventions pour les invocations de services à distance et pour les messages alors échangés.
- WSDL (*Web Services Description Language*) : langage basé sur XML permettant de décrire les *Web Services*.
- WS-Inspection : permet de localiser les descriptions des services publiés par un fournisseur de services.

Pour mieux comprendre comment fonctionnent les *Web Services*, nous allons traiter un exemple simple d'invocation de service à distance [Sotomayor, 2004]. Les différentes étapes d'une telle opération sont :

1. Le client lance une requête UDDI (*Universal Description, Discovery and Integration*) vers un *UDDI Registry* pour localiser les serveurs fournissant le service désiré.
2. L'*UDDI Registry* répond et indique au client où trouver ces serveurs.
3. Le client sait où trouver le serveur, mais il ne sait pas comment l'invoquer. Il lance donc une requête de description au serveur. Cette requête est effectuée en WSDL.
4. Le serveur répond, et indique au client comment invoquer ses services.
5. Le client peut maintenant invoquer le service. Il lance ainsi une requête SOAP à destination du serveur.
6. Le serveur reçoit cette requête, la traite, et renvoie au client une réponse SOAP.

Toutes ces étapes peuvent être résumées par le schéma de la figure 2.6.

Les *Grid Services* dérivant des *Web Services*, ils possèdent tous ces mécanismes permettant d'obtenir la description d'un service, ou bien d'invoquer un service à distance. Quelques fonctionnalités supplémentaires, nécessaires à la construction d'applications complexes destinées aux grilles de calcul, leur ont été rajoutées. Parmi celles-ci figurent la gestion du cycle de vie du service et le système de notification permettant à une entité de la grille d'être tenue informée des changements d'état d'un service.

[1] Les *Web Services* sont des services qui peuvent être invoqués par l'intermédiaire d'Internet. Un client peut lancer une requête de service à destination d'un serveur distant via le Web, et récupérer une réponse par le même intermédiaire.

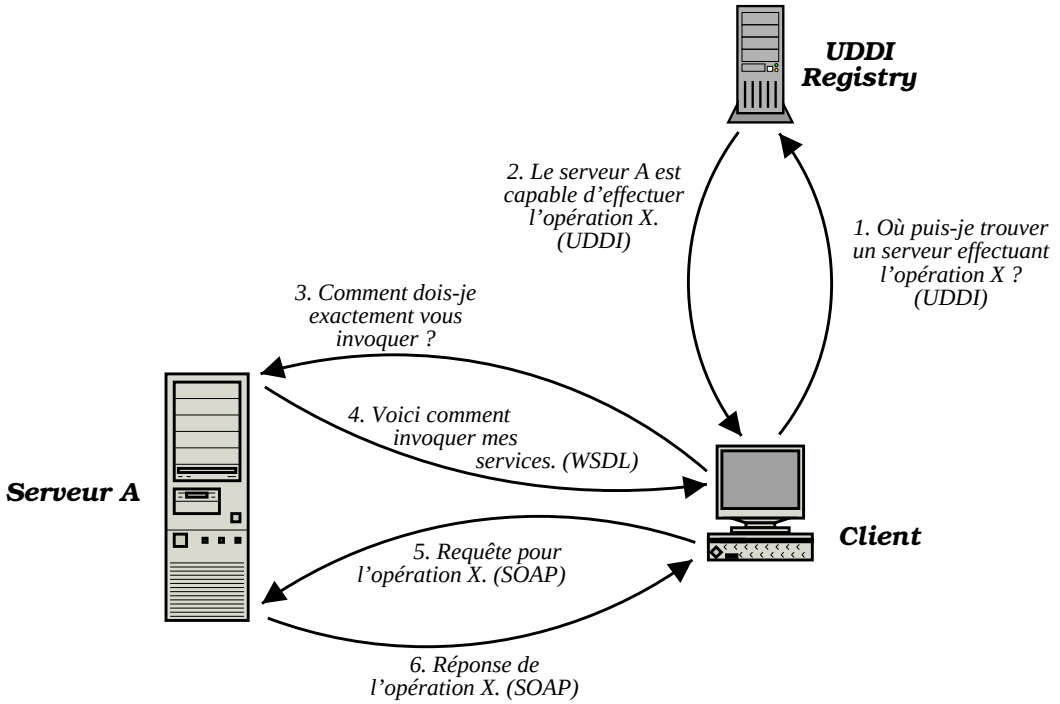


FIG. 2.6: Principe de l'invocation d'un Web Service

Cependant, la différence majeure qui oppose les *Grid Services* aux *Web Services* vient du fait que les *Web Services* sont des services persistants, qui survivent à leurs clients. Ils ne possèdent pas d'état, c'est-à-dire qu'ils sont incapables de se souvenir de leurs interactions avec les différents clients. Les *Grid Services* doivent, quant à eux, pouvoir être temporaires : un client peut demander la création d'un service, puis, lorsqu'il n'en a plus besoin, sa destruction. De plus, les *Grid Services* possèdent des états qui leur permettent de se souvenir de leurs interactions avec les clients, ce qui peut s'avérer extrêmement utile pour ces derniers.

2.4 Conclusion

Le projet Globus a permis d'aboutir à la création d'un intergiciel répondant à tous les problèmes fondamentaux qui peuvent se poser dans des systèmes tels que les grilles de calcul. Il permet la gestion des ressources d'une grille, la collecte et la consultation d'informations caractérisant l'état de la grille et des ressources qui la composent, ainsi que l'accès aux données présentes sur cette grille. De plus, tous les mécanismes de sécurité nécessaires sont également fournis. La norme OGSA permet de faire en sorte que l'accès aux services d'une grille soit aussi transparent que possible. Le concept des *Grid Services* amené par OGSA, favorise le rapprochement des technologies mises en œuvre ici avec celles déjà en vogue sur Internet (les *Web Services*).

Si la présence d'un intergiciel tel que Globus est nécessaire à l'élaboration d'une grille et à son maintien, l'existence d'une politique de gestion des ressources adéquate est indispensable à sa viabilité à long terme. En effet, une grille étant une agrégation de ressources mises à disposition d'autres utilisateurs, chacun de leurs propriétaires doit trouver une contrepartie à laisser d'autres personnes utiliser ses ressources. Les modèles économiques apportent une réponse à ce problème.

Chapitre 3

Modèles économiques pour la gestion des ressources d'une grille de calcul

Nous présenterons, dans ce chapitre, la manière dont les modèles économiques issus du marché réel peuvent servir de base pour la gestion des ressources d'une grille de calcul [Buyya, 2002]. Dans un premier temps, nous évaluerons pourquoi de tels modèles sont nécessaires, et quels sont leurs objectifs. Ensuite, nous étudierons le fonctionnement général de ces modèles. Nous pourrons alors analyser un par un les différents modèles économiques existants. Enfin, dans une dernière partie, nous approfondirons quelques aspects des modèles économiques, tels que les ressources pouvant être facturées, comment établir leur prix, et les mécanismes de paiement mis en jeu.

3.1 Introduction

Les grilles de calcul ont pour but de fédérer un nombre important de ressources géographiquement distribuées et d'offrir leur utilisation aux clients de la grille de la manière la plus transparente possible. Un tel environnement contient des ressources hétérogènes, du point de vue de leur nature, mais aussi des organisations qui les possèdent. Ces dernières ont leur propre politique de gestion des ressources, ainsi que des modèles de coût et d'utilisation pour différents utilisateurs à différents moments. La disponibilité des ressources et leur charge peuvent également fluctuer au cours du temps.

Dans un tel contexte, où clients et serveurs ont chacun leurs propres objectifs et stratégies, il est nécessaire de mettre en place des politiques de gestion des ressources qui permettent de contenter chacune des parties. Les approches traditionnelles dans ce domaine ne peuvent pas être employées. Elles utilisent des politiques de gestion centralisées qui nécessitent une connaissance de l'état de l'ensemble de la grille. Dans le cadre du *grid computing* tel que nous l'avons défini, il est impossible de détenir de telles informations.

Une approche basée sur les modèles économiques du marché réel peut être employée dans le cadre du calcul sur grille. Ces modèles économiques constitueront une métaphore pour la gestion des ressources et le placement d'applications, permettant ainsi de réguler l'offre et la demande au niveau des ressources de la grille tout en garantissant une certaine qualité de service aux clients.

L'objectif des modèles économiques est de permettre à la fois aux producteurs (les propriétaires des ressources) et aux consommateurs (les utilisateurs des ressources) d'atteindre leurs objectifs. Ils doivent ainsi inciter les producteurs à partager leurs ressources, en leur fournissant une contrepartie, et également inciter les consommateurs à réfléchir aux compromis entre temps de calcul et coût dépendant de la qualité de service désirée.

Ainsi, un gestionnaire de ressources basé sur l'utilisation des modèles économiques aura les avantages suivants :

- il incitera les possesseurs de ressources à autoriser leur utilisation durant leurs périodes d'inactivité,
- il permettra la régulation de l'offre et de la demande pour les ressources,
- il supprime le besoin d'un agent central durant les phases de négociation,
- il offre un traitement uniforme pour toutes les ressources, quelle que soit leur nature,
- il permet le développement de politiques d'ordonnancement basées sur les utilisateurs, et non sur le système,
- il place le pouvoir de décision dans les mains des producteurs et des consommateurs, qui sont libres d'effectuer leurs propres choix pour optimiser leurs profits respectifs.

3.2 Fonctionnement général

Les modèles économiques offrent un moyen de gérer les ressources d'une grille, en mettant en relation leurs propriétaires et leurs utilisateurs potentiels. Les producteurs seront rémunérés à chaque fois qu'un consommateur utilise leurs ressources. Pour cela, il existe deux grands modèles : un modèle basé sur l'échange, et un autre sur les prix. Dans le premier modèle, à chaque fois qu'une ressource est employée, son propriétaire acquiert des droits de consommation. Il pourra ainsi, lorsqu'il en aura besoin, utiliser à son tour les ressources de la grille. Cette façon de voir les choses suppose que tous les participants possèdent des ressources, ce qui n'est pas toujours le cas. Le second modèle permet aux consommateurs d'utiliser des ressources même s'ils n'en possèdent pas. Ils devront alors s'acquitter d'un prix à payer dépendant de la ressource utilisée.

Il est à noter que producteurs et consommateurs ont chacun leurs propres objectifs. Les propriétaires des ressources cherchent à maximiser leurs bénéfices, tandis que les utilisateurs désirent ne pas payer le prix fort, mais plutôt négocier un coût dépendant des caractéristiques de l'opération à effectuer, de leur budget, et de la qualité de service requise. L'établissement du prix d'une ressource dépendra aussi de l'offre et de la demande. Ainsi, dans cette approche, il est important de remarquer que chaque consommateur est en compétition avec les autres pour obtenir le droit d'utiliser une ressource, et chaque producteur est lui aussi en compétition avec les autres afin de remporter le marché.

Une grille utilisant les modèles économiques pour la gestion de ses ressources doit comprendre les éléments suivants :

- un annuaire d'information répertoriant les différentes entités de la grille,
- des modèles permettant d'établir la valeur d'une ressource,
- des mécanismes générant le prix des ressources et les publiant auprès de l'annuaire d'information,
- des protocoles de négociation entre producteurs et consommateurs,
- des agents intermédiaires, agissant en tant qu'agents de régulation, chargés d'établir la valeur des ressources à partir de modèles de prix, de gérer les échanges monétaires, et de gérer au mieux les situations de crise,
- des mécanismes de facturation et de paiement,
- des systèmes d'ordonnancement garantissant le respect de la qualité de service demandée pour les applications qui leur sont confiées.

Pour répondre aux spécifications ci-dessus, plusieurs entités doivent être mises en place :

- un courtier, ou *Grid Resource Broker* (GRB). Il agit comme un médiateur entre l'utilisateur et les ressources. Il est responsable :
  - de la découverte et de la sélection des ressources,

  - du transport des données et des programmes,
  - d'initier les exécutions sur les ressources distantes,
  - de retourner les résultats aux utilisateurs,
  - de s'adapter aux changements au niveau des ressources de la grille,
  - de présenter la grille à l'utilisateur comme étant une seule ressource unifiée.

  Il est à noter que le courtier représente l'utilisateur. Il connaît donc parfaitement ses objectifs, et tente de les satisfaire au mieux lors des négociations avec les fournisseurs de ressources.
- un annuaire, ou *Grid Market Directory* (GMD). Il permet aux propriétaires de ressources de publier leurs tarifs afin d'attirer des clients.
- un serveur d'échange, ou *Grid Trade Server* (GTS). Il s'agit d'un agent associé à un possesseur de ressources, et qui est chargé de négocier avec les clients dans le but de vendre l'accès aux ressources.
- un système de comptabilité. Il est responsable de suivre l'usage des ressources, et de facturer le client selon l'accord préalablement conclu entre le GRB et le GTS. Chaque producteur peut posséder son propre système de comptabilité, ou bien il peut en exister un global (*Grid Bank*).

3.3 Les différents modèles économiques existants

Les modèles économiques que nous décrirons successivement sont au nombre de cinq :

- le modèle de marché,
- le modèle des négociations,
- le modèle de l'appel d'offres,
- le modèle de la vente aux enchères,
- le modèle du partage proportionnel des ressources.

3.3.1 Modèle de marché

Dans ce modèle, les fournisseurs de ressources fixent un prix, et les utilisateurs sont facturés à ce prix, proportionnellement à la quantité de ressources consommées. Le prix décidé par les propriétaires de ressources peut être fixe, ou bien varier au cours du temps selon la loi de l'offre et de la demande.

La figure 3.1 illustre le principe de ce modèle.

1. Chaque fournisseur fixe un prix pour ses ressources et le publie auprès de l'annuaire.
2. Le courtier contacte l'annuaire pour obtenir la liste des fournisseurs avec leurs prix.
3. Le courtier identifie le fournisseur qui satisfait au mieux les objectifs du client, et lui confie l'application à exécuter.
4. Le fournisseur facture le client selon la quantité de ressources consommées.

Ce modèle peut être étendu par un système de publication d'offres promotionnelles ayant pour but de fidéliser les clients ou bien d'en attirer de nouveaux.

3.3.2 Modèle des négociations

Ce modèle permet à l'utilisateur, contrairement au précédent, d'agir sur le prix des ressources qu'il désire consommer. Ici, producteurs et consommateurs auront leurs propres objectifs, et devront négocier pour les atteindre.

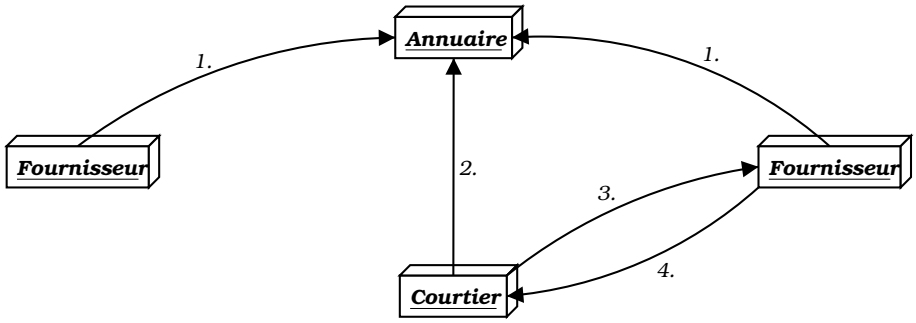


FIG. 3.1: Principe du modèle de marché

La figure 3.2 illustre le principe d'un tel modèle :

1. Chaque fournisseur s'enregistre auprès de l'annuaire.
2. Le courtier contacte l'annuaire pour obtenir la liste des fournisseurs.
3. Le courtier négocie avec un fournisseur pour obtenir un prix acceptable, le courtier commençant avec un prix faible, et le fournisseur avec un prix fort.
4. Si les deux parties se mettent d'accord sur un prix, l'application est confiée au fournisseur qui pourra ensuite facturer le client. Si aucun terrain d'entente ne peut être trouvé, le courtier s'adresse à un autre fournisseur, et entame une nouvelle négociation.

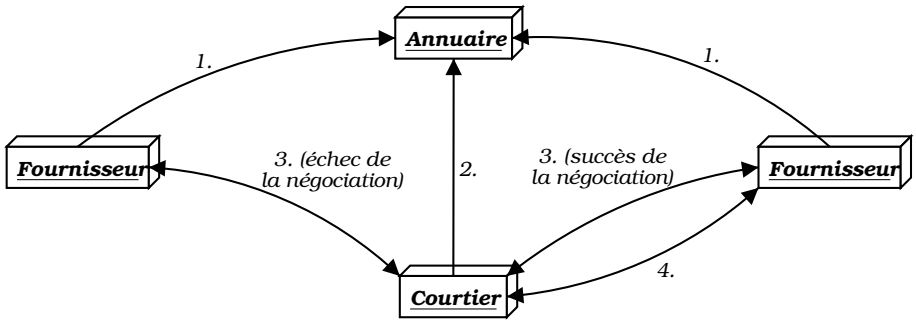


FIG. 3.2: Principe du modèle des négociations

Il est à noter que, grâce à cette approche, un courtier peut prendre des risques pour négocier des prix très bas, et éliminer les ressources trop chères. Ceci peut aboutir à une sous-utilisation des ressources, obligeant ainsi les fournisseurs à réduire leurs tarifs plutôt que de gaspiller des cycles d'utilisation de leurs ressources.

3.3.3 Modèle de l'appel d'offres

Le modèle de l'appel d'offres est inspiré des mécanismes mis en place dans le monde des affaires pour gouverner les échanges de biens et de services. Il permet au client de trouver le fournisseur de service approprié pour travailler sur une tâche donnée. Dans ce contexte, une nomenclature particulière est adoptée. Le client (ou le courtier qui le représente) est appelé « manager », tandis que chaque fournisseur est un « contractant » potentiel.

La figure 3.3 illustre le principe de ce modèle :

1. Chaque fournisseur s'enregistre auprès de l'annuaire.
2. Le manager contacte l'annuaire pour obtenir la liste des fournisseurs (contractants potentiels).

3. Le manager lance un appel d'offres auprès des contractants potentiels, en leur fournissant ses exigences.
4. Chaque contractant potentiel l'évalue, et lui soumet une offre s'il est intéressé.
5. Le manager choisit la meilleure offre et lui attribue le contrat.

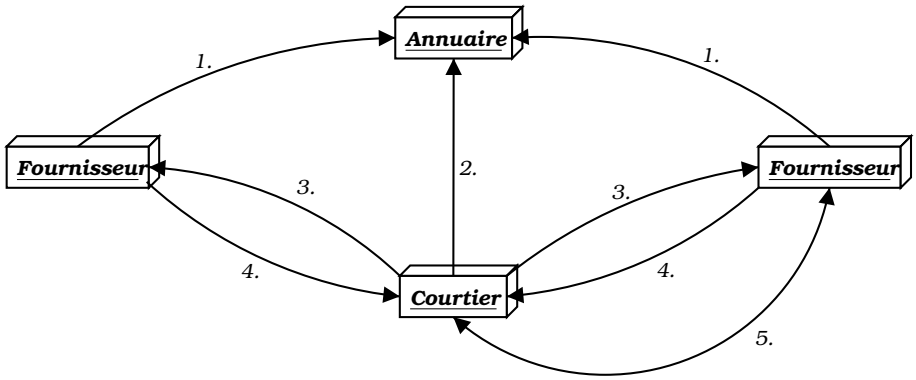


FIG. 3.3: Principe du modèle de l'appel d'offres

3.3.4 Modèle de la vente aux enchères

Ce modèle économique permet d'avoir une négociation entre un fournisseur et plusieurs consommateurs potentiels, qui permet d'aboutir à l'établissement d'un prix et au choix d'un consommateur. Une troisième entité entre également en jeu. Il s'agit du commissaire-priseur dont le rôle est de contrôler le bon déroulement de l'enchère.

La figure 3.4 illustre le principe du modèle des enchères.

1. Le fournisseur publie ses services auprès du commissaire-priseur et invite les utilisateurs à faire des offres.
2. Les différents courtiers font des offres, et peuvent surenchérir.
3. L'enchère s'arrête lorsque plus aucun courtier ne surenchérit. Si le prix minimum fixé par le fournisseur est atteint, la ressource est attribuée à l'utilisateur le plus offrant.

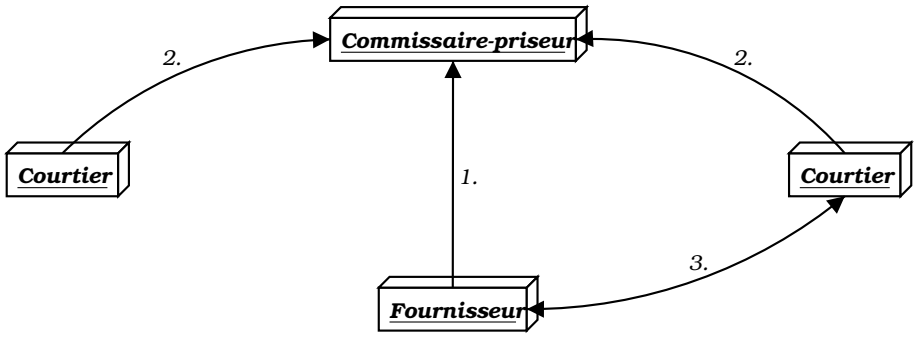


FIG. 3.4: Principe du modèle de la vente aux enchères

Si le type d'enchère décrit ci-dessus est le plus courant, il en existe trois variantes qui peuvent influer sur les stratégies à adopter de la part des courtiers :

- Chaque courtier soumet une et une seule offre sans connaître celles des autres. La meilleure offre l'emporte.
- Le commissaire-priseur part d'un prix très élevé, et le diminue progressivement jusqu'à ce qu'un courtier accepte le prix, ou bien jusqu'à ce que le prix atteigne une valeur seuil minimale.
- Les courtiers et les fournisseurs soumettent à tout moment des offres d'achat ou des ordres de vente. Si à un moment, il y a correspondance entre les deux, un accord est automatiquement passé entre les deux parties.

3.3.5 Modèle du partage proportionnel des ressources

Selon ce modèle, une ressource peut être partagée entre plusieurs utilisateurs. La part de ressource allouée à chaque utilisateur par rapport aux autres est alors proportionnelle à son offre par rapport à celles des autres utilisateurs. Ceci permet à tous les utilisateurs d'avoir accès aux ressources, même s'ils n'ont pas un budget important par rapport aux autres utilisateurs.

3.4 Etablissement des prix et mécanismes de paiement

Jusqu'à maintenant, nous avons vu comment des modèles économiques peuvent servir pour gérer au mieux les ressources constituant une grille. Dans la section précédente, nous avons détaillé les différents protocoles d'attribution d'une ressource à un utilisateur. Nous avons ainsi mis en avant deux aspects importants des modèles économiques : le prix des ressources (sur lequel repose leur attribution), et la facturation.

Dans cette section, nous allons étudier quelles sont les ressources qui peuvent être facturées, comment établir leur prix, et ensuite quels sont les mécanismes de paiement qui entrent en jeu lors de la facturation.

3.4.1 Ressources à facturer

Les ressources suivantes pourront être facturées à leurs utilisateurs :

- le temps processeur consommé,
- la quantité de mémoire consommée,
- la quantité de stockage utilisée,
- la bande-passante du réseau consommée,
- les signaux reçus et les changements de contexte,
- les logiciels et librairies utilisés.

3.4.2 Etablissement du prix d'une ressource

Selon le modèle économique que l'on désire appliquer à la gestion des ressources d'une grille, il sera nécessaire de leur donner un prix. Dans les modèles les plus simples, ce prix pourra être fixe. Mais ceci ne reflète pas la réalité. En effet, la valeur d'une ressource variera toujours au cours du temps, et ce en fonction du contexte dans lequel elle se trouve à un instant donné.

Ainsi, le prix d'une ressource dépend des paramètres suivants :

- la durée d'utilisation,
- le moment d'utilisation (notion d'heures pleines et d'heures creuses),
- l'offre et la demande,
- la fidélité de l'utilisateur,

- l'existence d'un pré-contrat entre l'utilisateur et le fournisseur,
- la puissance de la ressource,
- son coût physique,
- la surcharge du service,
- la valeur perçue par l'utilisateur,
- les préférences locales du propriétaire.

3.4.3 Mécanismes de paiement

La facturation est un des aspects fondamentaux des modèles économiques, car c'est sur lui que repose la viabilité de ce système. La notion de paiement suppose qu'il existe une ou plusieurs entités capables de gérer les comptabilités des fournisseurs. Il peut en exister une par fournisseur (chaque fournisseur facture directement ses clients et gère les mécanismes de paiement), ou bien une banque centralisant les comptabilités de chaque fournisseur peut être mise en place.

Le paiement pour l'utilisation de ressources peut s'effectuer à plusieurs moments :

- paiement à l'avance : l'utilisateur achète au préalable des crédits qu'il peut ensuite dépenser pour utiliser les ressources.
- paiement après l'utilisation : le client est facturé conformément à sa consommation de ressources.
- paiement au fur et à mesure de la consommation.

Le paiement peut s'effectuer de différentes manières :

- par chèque électronique,
- par monnaie électronique (identique au chèque, l'anonymat en plus),
- par carte de crédit,
- avec une monnaie virtuelle. Ce moyen de paiement est souvent utilisé lorsque tous les utilisateurs d'une grille possèdent des ressources (modèle basé sur l'échange). Lorsqu'une ressource est consommée, son propriétaire gagne un certain montant de cette monnaie virtuelle qu'il pourra utiliser plus tard pour consommer à son tour des ressources.

3.5 Conclusion

Dans ce chapitre, nous avons vu que les modèles économiques du marché réel peuvent s'appliquer au domaine du *grid computing*, dans le but de gérer au mieux les ressources présentes sur une grille. Par rapport aux modèles classiques de gestion de ressources, les modèles économiques ne nécessitent pas de contrôle centralisé, et de plus, ils ont pour priorité de prendre en compte les désirs des utilisateurs de la grille.

Dans le chapitre suivant, nous verrons comment appliquer les modèles économiques à l'outil Aroma, en cours de développement au LAAS.

Chapitre 4

Mise en œuvre d'un modèle économique simple

L'objectif de ce chapitre est de mettre en place un modèle économique simple. Pour cela, nous utiliserons l'outil Aroma, actuellement en cours de développement au LAAS. Après une brève présentation d'Aroma, nous effectuerons une étude théorique sur le modèle économique à mettre en œuvre. Nous verrons alors comment il est possible d'intégrer ce modèle au sein d'Aroma.

4.1 Présentation du contexte

4.1.1 Le projet CASP et Aroma

Le projet CASP (*Clusters for Application Service Provider*) [Gayola & al., 2002] vise la création de services Internet déportés sur une machine parallèle de type *cluster* pour des applications industrielles nécessitant d'importantes ressources informatiques. L'ambition de ce projet est de permettre aux entreprises d'exécuter leurs applications sur un *cluster* via une interface Internet personnalisable.

La mise en place de ce système passe par la construction de l'infrastructure logicielle nécessaire à la gestion du *cluster* pour réaliser de l'ASP (*Application Service Provider*) :

- administration du *cluster*,
- gestion des ressources (avec garantie de qualité de service),
- portail Internet.

Le principe d'utilisation de cette infrastructure est simple. Un utilisateur dispose sur son poste d'applications clientes. Ces dernières peuvent prendre en charge le travail à effectuer localement, dans le cas de tâches légères. Elles peuvent également faire appel à des services déportés pour réaliser des calculs plus coûteux. Dans ce cas, une demande est formulée via le client du gestionnaire de ressources, puis elle est transférée au fournisseur de services à travers des réseaux tels que l'Internet (figure 4.1). Cette transmission se fait en utilisant des mécanismes d'authentification et de confidentialité des informations émises.

La requête de demande de ressources est alors transmise au gestionnaire de ressources du site du fournisseur. Ce dernier doit garantir une efficacité d'utilisation des ressources de calcul, de la mémoire et des réseaux afin de satisfaire une qualité de service prédéfinie. Le gestionnaire de ressources est alors chargé d'exécuter la demande de l'utilisateur en la transmettant au logiciel applicatif serveur adéquat.

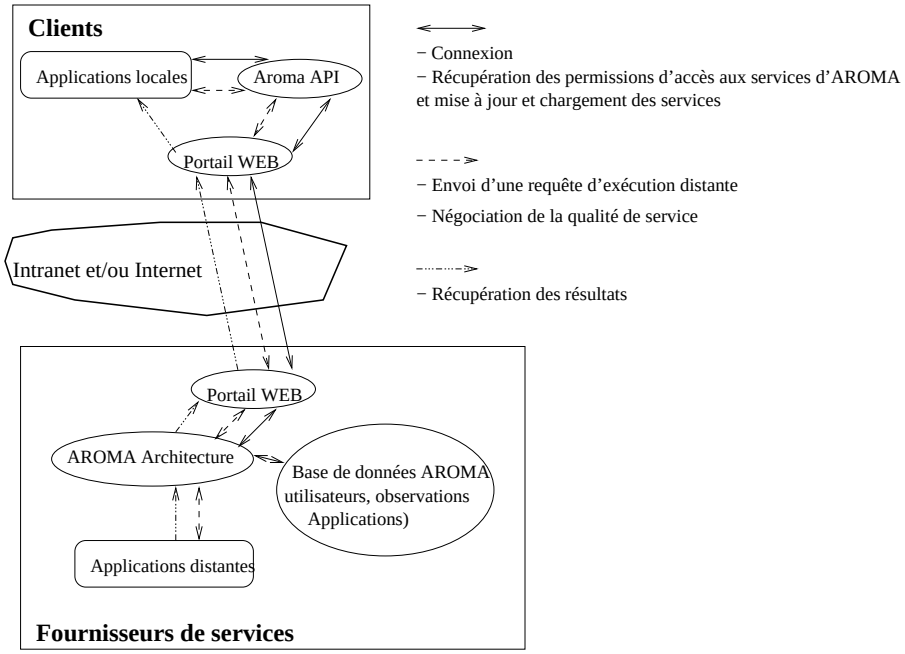


FIG. 4.1: Communications entre les fournisseurs de services et les clients

Aroma (*scAlable ResOurces Manager and wAtcher for application service provider*) [Pascal & al., 2002] correspond au gestionnaire de ressources du projet CASP. Il offre deux types d'outils principaux concernant :

- *Le placement de tâches et la gestion des ressources disponibles :* Les tâches sont placées sur des machines choisies par un algorithme de placement en fonction de leur état (taux d'utilisation du processeur, de la mémoire, etc.). Ce placement est soumis à certaines contraintes, notamment celle de garantir un certain niveau de qualité de service vis-à-vis de l'utilisateur.
- *L'observation des machines et du réseau :* Toutes les données concernant l'utilisation des machines (nombre de processus, taux d'utilisation du processeur et de la mémoire, etc.), ou bien la charge du réseau sont collectées et sauvegardées dans une base de données. Elles peuvent ensuite être traitées de manière à générer des statistiques exploitables par les administrateurs systèmes.

4.1.2 Objectifs fixés

L'objectif de ce chapitre est de répondre à une question simple : est-il possible d'intégrer la notion de modèles économiques au sein d'Aroma ? Cet outil, de par sa définition, constitue un intergiciel à part entière. L'ordonnanceur actuellement utilisé ne prend en compte que l'état des machines qu'il contrôle pour placer des tâches. A terme, l'objectif est d'utiliser les modèles économiques afin de gérer les ressources dont dispose Aroma.

La création d'un ordonnanceur basé sur les modèles économiques n'est pas un travail aisé. Ainsi, dans le cadre du stage de DEA, nous ne pourrons pas créer un tel ordonnanceur. Nous limiterons nos recherches à deux objectifs essentiels :

- Créer un modèle économique simple. Grâce à ce modèle, nous pourrons étudier, de manière théorique, le comportement d'un ordonnanceur basé sur ce modèle. Nous pourrons ainsi conclure sur la viabilité d'une approche orientée sur les modèles économiques pour la gestion de ressources.
- Modifier l'ordonnanceur actuel afin de prendre en compte les aspects liés à la facturation. Nous conserverons l'algorithme de placement utilisé, mais nous y inclurons des mécanismes de décompte des ressources consommées qui nous permettront d'établir un prix, à posteriori, pour l'exécution d'une application.

4.2 Etude théorique

Dans cette section, nous développerons un modèle économique permettant la gestion des ressources d'une grille. Dans un premier temps, nous présenterons les caractéristiques principales de notre modèle. Ensuite, nous définirons les variables que nous utiliserons dans ce modèle. Nous pourrons alors présenter les équations le régissant. Enfin, nous réaliserons une étude sur le comportement théorique d'un ordonnanceur utilisant ce modèle économique pour attribuer des ressources à une application.

4.2.1 Caractéristiques du modèle économique

Dans le chapitre 3, nous avons décrit les différents modèles économiques existants, ainsi que les ressources qu'ils pouvaient prendre en compte. Le modèle que nous allons étudier ici se rapproche du modèle économique de marché (section 3.3.1). Le prix à payer, après l'exécution d'une application, sera proportionnel à la quantité de ressources consommées, le prix unitaire étant fixe et connu à l'avance.

Les ressources que nous prendrons en compte dans notre modèle sont celles concernant les processeurs des machines utilisées. Une application devant être exécutée sera tout d'abord découpée en plusieurs tâches. Nous supposerons ces tâches identiques du point de vue des ressources qu'elles consomment. Chaque tâche occupera un et un seul processeur (chaque processeur disponible pourra accueillir plusieurs tâches), pendant un temps donné. Notons que, même si toutes les tâches d'une application sont identiques, le temps pendant lequel elles seront exécutées pourra varier d'une tâche à l'autre en fonction de la puissance des processeurs qu'elles occupent.

Dans un tel contexte, nous proposons de facturer une application selon les ressources suivantes :

- le temps processeur consommé : il s'agit de la somme du temps réel pendant lequel chaque tâche de l'application aura occupé un processeur,
- le nombre de processeurs utilisés,
- la puissance de chaque processeur utilisé.

Ainsi, selon le coût de chacune de ces ressources, différentes stratégies pourront être adoptées pour minimiser le coût global de l'application. Le problème consistera donc à choisir les machines (leur nombre et leur puissance), de telle sorte que le placement des tâches sur celles-ci offre un bon compromis entre le coût de l'application et sa durée d'exécution.

4.2.2 Définition des variables

Dans cette section, nous allons présenter les variables permettant de traiter correctement notre problème. Notons qu'elles sont réparties en deux grandes catégories : les données du problème, et les inconnues, ces dernières fournissant l'ordonnancement trouvé.

Les données du problème

P^R = puissance du processeur de référence. Lorsque l'utilisateur effectue une requête, il doit renseigner la durée totale de l'application, en prenant pour référence cette puissance.

T^R = temps de référence (temps global de l'application demandé par l'utilisateur).

A = nombre de tâches de l'application.

N^P = nombre de processeurs disponibles.

N^P_{min} = borne minimale imposée par l'utilisateur sur le nombre de processeurs à utiliser.

N^P_{max} = borne maximale imposée par l'utilisateur sur le nombre de processeurs à utiliser.

$P(j)_{1 \leq j \leq N^P}$ = puissance de chaque processeur disponible.

$Ch(j)_{1 \leq j \leq N^P}$ = charge de chaque processeur.

C^T = coût correspondant au temps processeur consommé.

C^N = coût correspondant au nombre de processeurs utilisés.

C^P = coût correspondant à la puissance des processeurs utilisés.

$C^P_{coef}(j)_{1 \leq j \leq N^P}$ = coefficient appliqué sur le coût de chaque processeur. Ce coefficient permet de modifier localement le coût d'un processeur particulier.

Les inconnues

$Taches(i,j)_{1 \leq i \leq A, 1 \leq j \leq N^P}$ = matrice associant chaque tâche de l'application à un des processeurs disponibles. Le résultat de l'ordonnancement est donc contenu dans cette matrice. Il s'agit ainsi de l'inconnue principale du problème. Toutes les inconnues suivantes sont déduites de cette matrice.

$U(j)_{1 \leq j \leq N^P}$ = vecteur indiquant, pour chaque processeur, s'il est utilisé ou non.

N = nombre de processeurs utilisés.

$y(j)_{1 \leq j \leq N^P}$ = variables d'écart entrant en jeu dans le calcul du vecteur U. Elles permettent d'avoir des résultats entiers dans ce vecteur.

$Temps(j)_{1 \leq j \leq N^P}$ = temps cumulé occupé par l'application sur chaque processeur. Il s'agit du temps passé réellement par les tâches sur chaque processeur (correspondant à la différence entre la date de fin de la dernière des tâches et la date de début de la première). Il dépend donc de la puissance de ces derniers, ainsi que de leur charge.

C_{App} = coût de l'application.

T_{App} = durée d'exécution de l'application.

Fonction objectif

La fonction objectif est un compromis entre le coût de l'application et sa durée d'exécution. Ces deux paramètres sont pondérés par des coefficients ajustables selon ce qu'il est souhaité de privilégier. La fonction objectif est donc la suivante :

$$Objectif = PoidsCout \times C_{App} + PoidsDuree \times T_{App}$$

Ainsi, il faudra calculer le contenu de la matrice $Taches$ de manière à minimiser cette fonction objectif.

Quelques remarques

Avant de poursuivre notre étude en posant les équations régissant notre modèle économique, nous allons apporter quelques précisions sur les variables précédentes et donner des exemples de leur utilisation.

Prenons le cas d'une application composée de 5 tâches ($A = 5$). L'utilisateur indique que cette application est exécutée en 100 secondes sur un processeur de puissance 200 ($T^R = 100$ et $P^R = 200$). En prenant pour hypothèse que chaque tâche de l'application est identique, leur durée sur le processeur de référence est de $T^R/A = 100/5 = 20$.

Pour exécuter cette tâche, 6 processeurs sont disponibles ($N^P = 6$). L'utilisateur spécifie qu'il veut en utiliser entre 3 et 5 ($N^P_{min} = 3$ et $N^P_{max} = 5$). La puissance et la charge des processeurs seront contenues dans les vecteurs suivants :

$$P = \begin{bmatrix} 200 \\ 200 \\ 400 \\ 600 \\ 600 \\ 800 \end{bmatrix} \quad Ch = \begin{bmatrix} 0.12 \\ 0.80 \\ 0.51 \\ 0.33 \\ 0.01 \\ 0.99 \end{bmatrix}$$

Notons que la charge d'un processeur s'exprime en pourcentage de processeur occupé.

Après minimisation de la fonction objectif dépendant de ces paramètres, nous obtiendrons des valeurs pour les inconnues. Tout d'abord, nous obtiendrons le contenu de la matrice $Taches$. Elle sera remplie comme suit :

- $Taches(i, j) = 1$ si la tâche i est placée sur le processeur j,
- $Taches(i, j) = 0$ sinon.

Par exemple,

$$Taches = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Nous voyons ici que le processeur 1 est associé à la tâche 4, le processeur 3 à la tâche 1, et enfin que le processeur 4 exécute les tâches 2, 3 et 5.

Nous pouvons alors déduire que 3 processeurs sont utilisés ($N = 3$). Nous pouvons également calculer le contenu du vecteur U, de la manière suivante :

- $U(j) = 1$ si le processeur j est utilisé,
- $U(j) = 0$ sinon.

Ainsi, on a :

$$U = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Notons enfin que le vecteur $Temps$, contenant la durée pendant laquelle les tâches de l'application auront occupé chaque processeur, sera calculé à partir du nombre de tâches attribué à chaque processeur, de la puissance de ces derniers ainsi que de leur charge. La formule sera donnée dans la section suivante. Cependant, nous pouvons établir que $Temps(2) = Temps(5) = Temps(6) = 0$, puisqu'aucune tâche n'est associée à ces deux processeurs.

4.2.3 Modèle mathématique

Dans cette section, nous allons établir toutes les équations et contraintes régissant notre modèle économique. Nous allons tout d'abord poser des contraintes sur le contenu de la matrice $Taches$. Ensuite, nous donnerons les équations permettant de calculer le vecteur U ainsi que le vecteur $Temps$ à partir de la matrice $Taches$. Enfin, nous donnerons les formules permettant de calculer le coût global de l'application ainsi que sa durée d'exécution.

Contraintes sur la matrice $Taches$

$$\forall i \in \{1, ..., A\} \qquad \forall j \in \{1, ..., N^P\} \qquad Taches(i,j) \in \{0,1\}$$

$$\forall i \in \{1, ..., A\} \qquad \sum_{j=1}^{N^P} Taches(i,j) = 1$$

Nous avons ainsi spécifié que chaque case de la matrice contient un 0 ou bien un 1. Nous avons également contraint chaque colonne à contenir un et un seul 1 (chaque tâche est placée sur un et un seul processeur).

Calcul du vecteur U

$$\forall j \in \{1, ..., N^P\} \qquad U(j) \in \{0,1\}$$

$$\forall j \in \{1, ..., N^P\} \qquad U(j) = \left[\left(\sum_{i=1}^{A} Taches(i,j) \right) + y(j) \right] / A$$

$$\forall j \in \{1, ..., N^P\} \qquad y(j) \in \mathbb{N} \qquad 0 \leq y(j) \leq A - 1$$

$U(j)$ est un entier égal à 1 si la ligne j de la matrice $Taches$ contient au moins un 1, ou égal à 0 sinon. Ainsi, nous introduisons une variable d'écart $y(j)$ dans le calcul de $U(j)$. Cette variable sera ajustée de telle sorte que $U(j)$ soit entier. On aura ainsi :

- $y(j) = 0$ si $\sum_{i=1}^{A} Taches(i,j) = 0 \qquad (\Rightarrow U(j) = 0)$,
- $y(j) = A - \sum_{i=1}^{A} Taches(i,j)$ sinon $\qquad (\Rightarrow U(j) = 1)$.

Calcul de N

$$N = \sum_{j=1}^{N^P} U(j)$$

$$0 \leq N^P_{min} \leq N \leq N^P_{max} \leq N^P$$

$$N^P_{max} \leq A$$

Nous voyons ainsi que plusieurs contraintes sur le nombre de processeurs utilisés entrent en jeu. Ce nombre ne doit pas franchir les bornes fixées par l'utilisateur. D'autre part, ces bornes doivent respecter les limites physiques du système. De plus, l'utilisateur ne peut pas demander plus de processeurs que ce qu'il y a de tâches à exécuter, puisque, dans ce cas-là, des processeurs ne seraient pas utilisés.

Calcul du vecteur $Temps$

Soit T le temps processeur consommé par une tâche sur un processeur de puissance P, on a :

$$T \cdot P = \frac{T^R \cdot P^R}{A} \qquad \Rightarrow \qquad T = \frac{T^R \cdot P^R}{A} \cdot \frac{1}{P}$$

Soit T_i le temps processeur consommé par la tâche i (lorsqu'elle est active), on a :

$$\forall i \in \{1, ..., A\} \qquad T_i = \frac{T^R \cdot P^R}{A} \cdot \sum_{j=1}^{N^P} \frac{Taches(i,j)}{P(j)}$$

Soit T_i^* le temps occupé par la tâche i sur un processeur (qu'elle soit active ou non), on a :

$$\forall i \in \{1, ..., A\} \qquad T_i^* = \frac{T^R \cdot P^R}{A} \cdot \sum_{j=1}^{N^P} \frac{Taches(i,j)}{P(j) \cdot \left(1 - Ch(j)\right)}$$

On en déduit le temps pendant lequel l'application occupera ainsi chaque processeur (que les tâches soient actives ou non) :

$$\forall j \in \{1, ..., N^P\} \qquad Temps(j) = \frac{T^R \cdot P^R}{A} \cdot \sum_{i=1}^{A} \frac{Taches(i,j)}{P(j) \cdot \left(1 - Ch(j)\right)}$$

Calcul du coût de l'application

$$C_{App} = C^T \cdot \frac{T^R \cdot P^R}{A} \cdot \sum_{i=1}^{A} \sum_{j=1}^{N^P} \frac{Taches(i,j)}{P(j)}$$

$$+C^N \cdot N$$

$$+C^P \cdot \sum_{i=1}^{A} \sum_{j=1}^{N^P} Taches(i,j) \cdot P(j) \cdot C_{coeff}^P(j)$$

Le coût de l'application dépend du temps processeur consommé par chaque tâche, du nombre de processeurs utilisés, ainsi que de leur puissance.

Calcul de la durée d'exécution de l'application

$$T_{App} = \max_{j=1,...,N^P} Temps(j)$$

Nous supposerons que les temps de communication et de synchronisation entre les tâches parallèles sont négligeables. Ceci est possible dans le cadre d'applications « gros grains », c'est-à-dire des applications dont les tâches calculent beaucoup et communiquent peu. Dans un tel contexte, la durée d'exécution de l'application correspond au maximum des temps qu'elle occupe sur chacun des processeurs.

Fonction objectif

Nous rappelons que la fonction à minimiser est la suivante :

$$Objectif = PoidsCout \times C_{App} + PoidsDuree \times T_{App}$$

Minimiser la fonction objectif revient donc à calculer le contenu de la matrice $Taches$ (la seule véritable inconnue du problème).

4.2.4 Résultats

Après avoir défini notre modèle économique, nous pouvons maintenant tester son comportement. Pour cela, nous allons prendre un exemple d'application, une liste de processeurs, et tenter de minimiser la fonction objectif pour différentes valeurs des paramètres de notre modèle. Pour effectuer cette minimisation, nous utiliserons le logiciel Xpress, un outil de modélisation et d'optimisation numérique, dont le but est de trouver la solution optimale d'un problème linéaire.

Définition de l'exemple traité

Nous traiterons ici l'exemple d'une application de vingt tâches ($A = 20$) à placer sur trente processeurs ($N^P = 30$). La puissance de chacun des processeurs est donnée dans le vecteur suivant :

$$P = \begin{bmatrix} 100 \\ 200 \\ 300 \\ \vdots \\ 2800 \\ 2900 \\ 3000 \end{bmatrix}$$

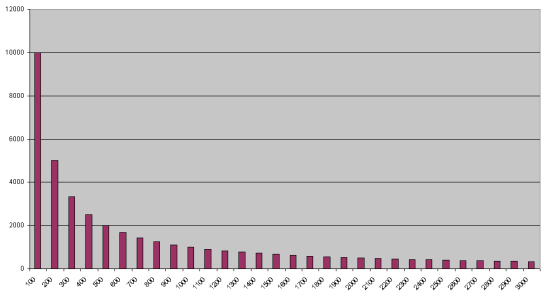
La puissance du processeur de référence est de 100 ($P^R = 100$). Le temps global de l'application demandé par l'utilisateur sur le processeur de référence est de 200000 ($T^R = 200000$). Notons enfin que l'utilisateur demande à utiliser entre 4 et 20 processeurs ($N^P_{min} = 4$ et $N^P_{max} = 20$).

Minimisation de la durée d'exécution de l'application

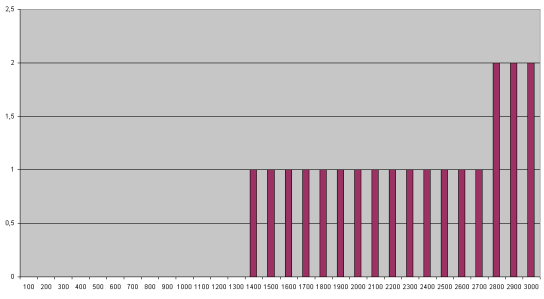
Dans un premier temps, nous allons tenter de minimiser la durée d'exécution de l'application, en trouvant l'ordonnancement optimal permettant d'exécuter les tâches au plus vite, quel que soit le prix à payer ($PoidsCout = 0$ et $PoidsDuree = 1$). Nous réaliserons cette étude avec les paramètres suivants :

$$C^T = 1 \qquad C^N = 1 \qquad C^P = 50$$

La figure 4.2 montre la répartition des tâches sur les différents processeurs afin d'obtenir un temps d'exécution optimal. Cette étude a été réalisée en supposant que la charge de chacun des processeurs était nulle.



(a) Temps d'exécution d'une tâche en fonction de la puissance des processeurs



(b) Ordonnancement choisi (nombre de tâches par processeur)

FIG. 4.2: Minimisation de la durée d'exécution de l'application

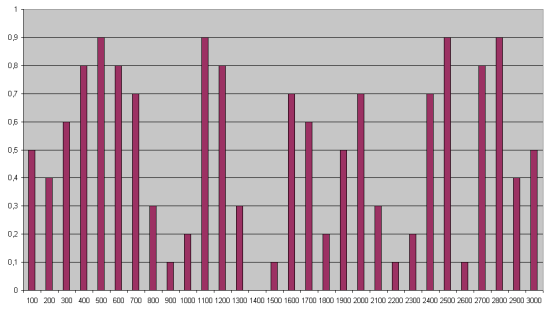
Nous constatons que ce sont les processeurs les plus rapides qui sont utilisés. De plus, le nombre de processeurs mis en jeu est assez important, et ce afin de ne pas les surcharger pour qu'ils puissent exécuter les tâches en un minimum de temps.

Notons que nous obtenons les valeurs suivantes :

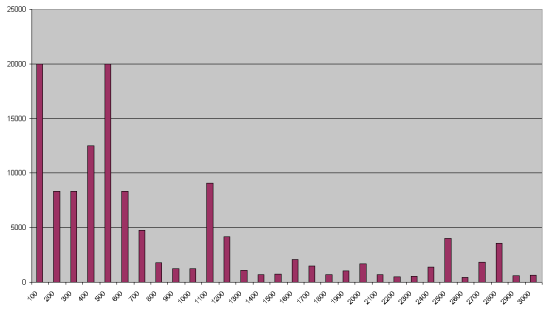
$$T_{app} = 714,286 \qquad C_{app} = 2314201,1 \qquad N = 17$$

La valeur de T_{app} est la durée minimale d'exécution de l'application. Il est totalement impossible, en conservant la configuration du système, de la faire diminuer. Notons également que le coût de l'application semble être très élevé.

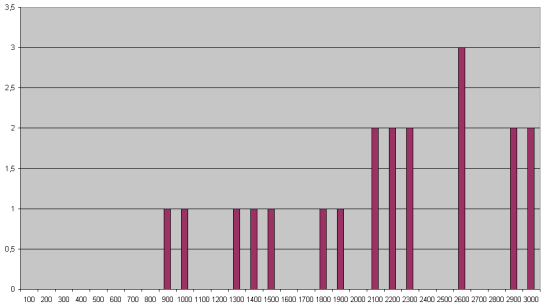
Essayons maintenant de minimiser la durée d'exécution de l'application dans le cas où les processeurs ont une charge initiale (figure 4.3).



(a) Charge de chaque processeur



(b) Temps d'exécution d'une tâche pour chaque processeur (prise en compte de la charge)



(c) Ordonnancement choisi (nombre de tâches par processeur)

FIG. 4.3: Minimisation de la durée d'exécution de l'application avec prise en compte de la charge des processeurs

Nous voyons ici que l'ordonnancement est affecté par la charge des machines. Même si leur puissance est élevée, nous constatons qu'aucune tâche n'est placée sur les machines très chargées. Au contraire, ce sont les machines peu chargées qui sont privilégiées, même si elles sont moins puissantes.

Les valeurs que nous obtenons maintenant sont :

$$T_{app} = 1360,544 \qquad C_{app} = 2140599,2 \qquad N = 13$$

Nous pouvons ainsi en conclure que la durée d'exécution de l'application a augmenté avec ce changement d'ordonnancement, ce qui était prévisible. Quant au coût de l'application, il reste très élevé.

Minimisation du coût de l'application

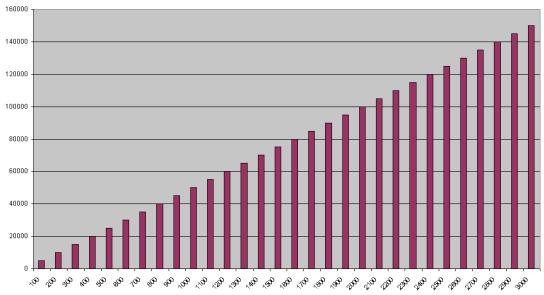
Nous allons maintenant tenter l'approche inverse, à savoir minimiser le coût de l'application sans tenir compte de sa durée d'exécution ($PoidsCout = 1$ et $PoidsDuree = 0$).

Nous allons commencer cette étude en mettant en avant les différences qu'il existe entre les résultats obtenus ici et les précédents. Pour cela, nous considérons que la charge initiale des différents processeurs est nulle. Il est à noter que nous gardons les mêmes paramètres de coût que précédemment, c'est-à-dire :

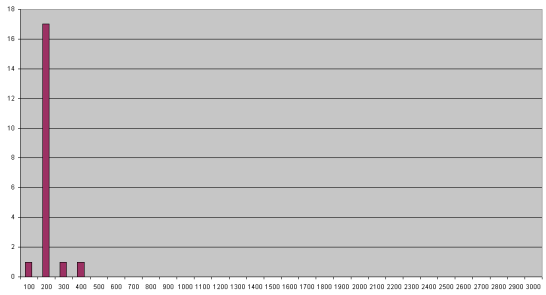
$$C^T = 1 \qquad C^N = 1 \qquad C^P = 50$$

Ces paramètres indiquent que les processeurs de forte puissance sont extrêmement coûteux. Notons que, pour l'instant, nous n'introduisons pas la notion des coefficients locaux applicables au coût de chacune des machines.

La figure 4.4 montre l'ordonnancement choisi pour ces paramètres.



(a) Coût de chaque processeur (proportionnel à la puissance)



(b) Ordonnancement choisi (nombre de tâches par processeur)

FIG. 4.4: Minimisation du coût de l'application (influence de la puissance des processeurs)

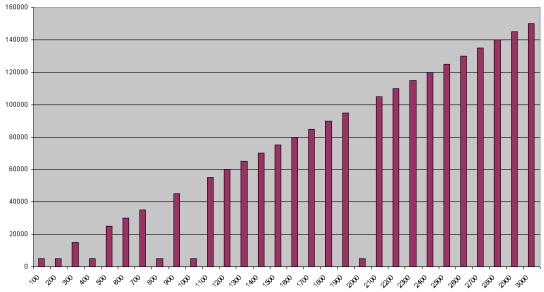
Nous observons que, contrairement à ce qu'il se passait précédemment, ce sont les machines de faible puissance qui sont choisies. Ceci prouve que le coût lié à la puissance est pris en compte pour l'ordonnancement des tâches de l'application.

Nous obtenons désormais les valeurs suivantes :

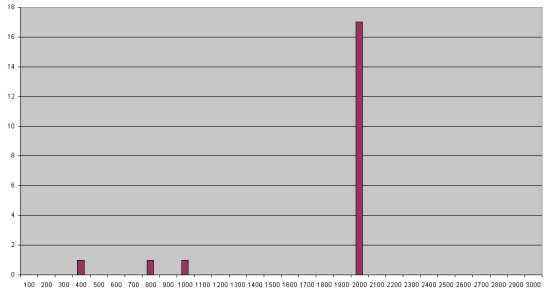
$$T_{app} = 85000 \qquad C_{app} = 310837,3 \qquad N = 4$$

Nous constatons que le coût de l'application a considérablement baissé. Nous avons ainsi atteint le prix minimal pour les paramètres de coût donnés. La durée d'exécution de l'application a, quant à elle, extrêmement augmenté. Ceci traduit le fait que notre ordonnanceur n'en tient pas du tout compte pour effectuer le placement de tâches. Il est également à noter que seulement quatre processeurs sont utilisés. Ceci est normal car le nombre de processeurs utilisés est payant, amenant ainsi l'ordonnanceur à en utiliser le moins possible ($N = N^P_{min}$).

La figure 4.5 démontre que, si nous conservons les mêmes paramètres de coût, mais que nous modifions localement le coût de certaines machines, ce sont les machines les moins chères qui sont privilégiées.



(a) Coût de chaque processeur (prise en compte des coefficients locaux)



(b) Ordonnancement choisi (nombre de tâches par processeur)

FIG. 4.5: Minimisation du coût de l'application (utilisation de coefficients de coût locaux)

Modifions maintenant les paramètres de coût, en mettant l'accent sur le prix du temps processeur consommé par les tâches :

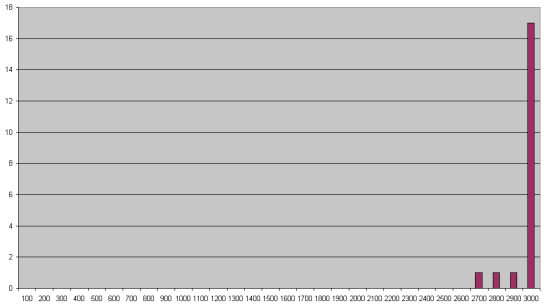
$$C^T = 50 \qquad C^N = 1 \qquad C^P = 1$$

La figure 4.6 nous donne l'ordonnancement choisi dans un tel cas.

Nous voyons, dans ce cas, que ce sont les processeurs de puissance élevée qui sont choisis, et ce en vue de minimiser le temps processeur consommé, et ainsi le coût global de l'application. Notons que seulement quatre processeurs sont utilisés, car le but n'est pas de minimiser la durée globale d'exécution de l'application, mais seulement le temps pendant lequel les tâches, prises individuellement, occupent activement un processeur.

Nous obtenons alors les valeurs suivantes :

$$T_{app} = 5666,661 \qquad C_{app} = 396354,9 \qquad N = 4$$



(a) Ordonnancement choisi (nombre de tâches par processeur)

FIG. 4.6: Minimisation du coût de l'application (influence du temps processeur consommé)

En les comparant avec les résultats précédents, nous voyons que le coût de l'application conserve le même ordre de grandeur. Il est minimal par rapport aux paramètres de coût que nous venons d'appliquer. Il est à noter que la durée d'exécution de l'application a considérablement chuté. Elle n'a cependant pas atteint sa valeur minimale.

Minimisation d'un compromis entre coût et durée d'exécution

Nous allons maintenant tenter de minimiser à la fois le coût de l'application et sa durée d'exécution. Nous devons donc ajuster les poids de ces deux valeurs dans la fonction objectif.

Les paramètres de coût que nous utiliserons sont les suivants :

$$C^T = 1 \qquad C^N = 1 \qquad C^P = 50$$

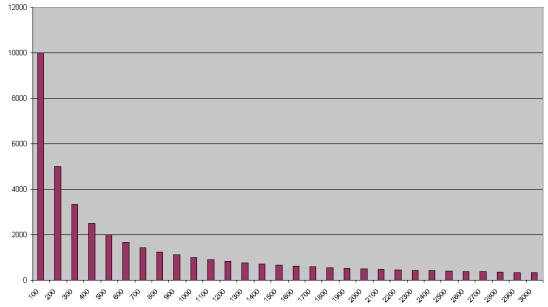
Nous savons que, pour de tels paramètres, nous avons :

$$C_{app_{min}} = 310837,3 \qquad T_{app_{min}} = 714,286$$

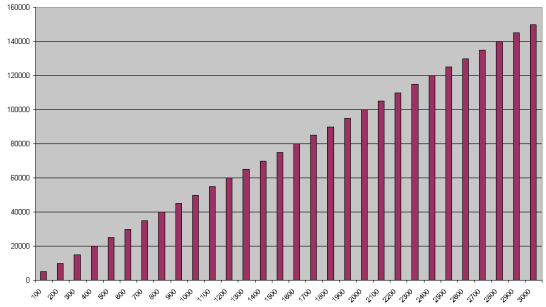
Pour trouver un bon compromis entre C_{app} et T_{app}, nous voyons qu'il faut multiplier T_{app} par environ 500. Ces deux valeurs auront ainsi le même ordre de grandeur. Nous avons donc :

$$PoidsCout = 1 \qquad PoidsDuree = 500$$

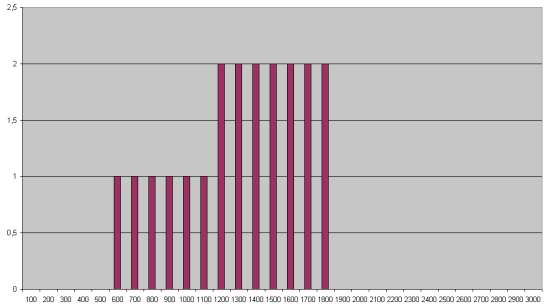
En prenant le cas où chaque processeur a une charge initiale nulle, et où aucun coefficient n'est appliqué sur le coût d'un processeur en particulier, nous obtenons la situation représentée par la figure 4.7.



(a) Temps d'exécution d'une tâche pour chaque processeur



(b) Coût de chaque processeur



(c) Ordonnancement choisi (nombre de tâches par processeur)

FIG. 4.7: Minimisation d'un compromis entre coût et durée d'exécution

Nous pouvons constater que le compromis choisi se répercute immédiatement sur les machines élues par l'ordonnanceur. En effet, il s'agit de machines comprises dans un domaine de puissances intermédiaires, permettant d'avoir une certaine rapidité d'exécution pour un coût assez faible.

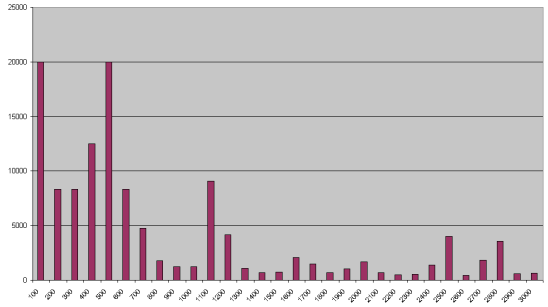
Dans ce cas, les valeurs que nous obtenons sont :

$$T_{app} = 1666.67 \qquad C_{app} = 1321882.981 \qquad N = 13$$

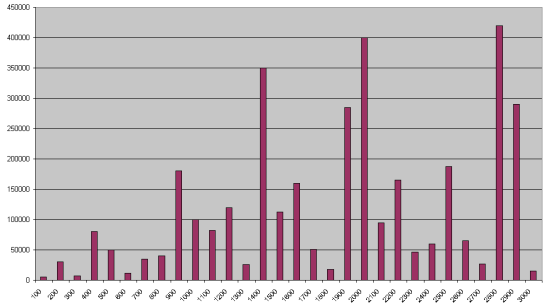
Nous voyons ici que ni le coût de l'application, ni sa durée d'exécution ne sont optimaux. Ceci est conforme aux résultats attendus, puisque nous voulions trouver un bon compromis entre les deux. De plus, le besoin de rapidité d'exécution force l'ordonnanceur à utiliser plus de processeurs, tout en essayant de ne pas faire trop augmenter le coût.

Pour compléter notre étude, nous allons maintenant prendre le cas de la minimisation d'un compromis entre le coût et la durée d'exécution de l'application, mais en prenant en compte la charge initiale de chaque machine, et les préférences locales en matière de prix d'un processeur.

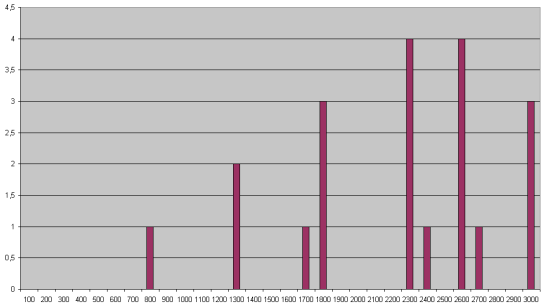
La figure 4.8 représente une telle situation.



(a) Temps d'exécution d'une tâche pour chaque processeur (prise en compte de la charge)



(b) Coût de chaque processeur (prise en compte des coefficients locaux)



(c) Ordonnancement choisi (nombre de tâches par processeur)

FIG. 4.8: Minimisation d'un compromis entre coût et durée d'exécution (cas complet)

Nous voyons que l'ordonnanceur doit composer avec la charge des machines dans le but de placer les tâches tout en minimisant la fonction objectif. Ainsi, les processeurs sont choisis en fonction de leur puissance, de leur charge et de leur coût.

Les valeurs que nous obtenons alors sont les suivantes :

$$T_{app} = 2197.8 \qquad C_{app} = 783117 \qquad N = 9$$

Comme nous pouvions nous y attendre, la durée d'exécution de l'application est plus élevée que dans le cas précédent, ce qui est expliqué par la charge des machines. Cependant, le coût de l'application a baissé. Ceci est normal puisque des « réductions » ont été appliquées sur le prix de certaines machines, par le biais des coefficients locaux.

4.2.5 Conclusion

Grâce au logiciel Xpress, nous avons pu simuler le comportement d'un ordonnanceur basé sur notre modèle économique. En effet, cet outil nous a permis de calculer, pour différents paramètres, l'ordonnancement qui permet de minimiser la fonction objectif de notre modèle.

Nous avons ainsi pu observer qu'un tel ordonnanceur possède le comportement attendu. Si la minimisation du coût de l'application est à privilégier, nous constatons que ce sont les machines les moins chères qui sont choisies. Si au contraire, la rapidité d'exécution est à mettre en avant, alors ce sont les machines les plus puissantes et les moins chargées qui sont choisies pour y placer des tâches. Enfin, dans une situation de compromis, l'ordonnanceur doit choisir des processeurs assez puissants, peu chargés, et bien sûr dont le prix reste raisonnable.

4.3 Mise en œuvre

Dans cette section, nous allons décrire comment nous avons mis en place un service de facturation au sein d'Aroma. En effet, après avoir démontré qu'un ordonnanceur basé sur les modèles économiques possède les caractéristiques que nous espérions, nous devons maintenant tenter de répondre à la question suivante : est-il possible, étant donné un ordonnanceur quelconque, d'y inclure des mécanismes de facturation ?

Pour cela, nous allons utiliser l'ordonnanceur actuel d'Aroma, et y effectuer les modification nécessaires afin de permettre d'établir un prix pour l'exécution d'une application.

Nous réaliserons cette étude en deux temps. Tout d'abord, nous mettrons en place une interface graphique permettant de sauvegarder, dans une base de données, les paramètres de coût (C^T, C^N, C^P et aussi C^P_{coeff}). Ensuite, nous pourrons modifier l'ordonnanceur afin qu'il facture toute application selon ces paramètres.

4.3.1 Sauvegarde des paramètres de coût dans une base de données

Avant de pouvoir facturer une application, il est indispensable de connaître les paramètres de coût à appliquer. Pour cela, nous choisissons de les stocker dans une base de données, par le biais d'une interface graphique. Cette interface nécessite d'être intégrée dans l'interface cliente d'Aroma afin que toute personne possédant des droits d'administration puisse y accéder.

L'interface à mettre en place doit permettre d'enregistrer deux types de paramètres :

- les paramètres communs à toutes les machines (C^T, C^N et C^P),
- les paramètres spécifiques à chacune des machines (C^P_{coeff}).

Pour traiter les paramètres issus de la première catégorie, nous mettons en place l'interface de la figure 4.9. Cette fenêtre permet à l'utilisateur de modifier C^T, C^N et C^P. Elle s'interface alors avec un serveur d'Aroma afin de lui demander de sauvegarder les nouvelles valeurs dans une base de données (figure 4.10).

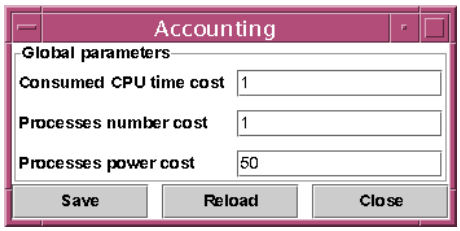


FIG. 4.9: Fenêtre permettant de fixer les paramètres de coût communs aux machines

Une table spécifique est alors mise en place dans la base de données. Son rôle est d'accueillir les valeurs de chacun des paramètres, indexées par la date de sauvegarde, afin de pouvoir conserver un historique des modifications de ces paramètres.

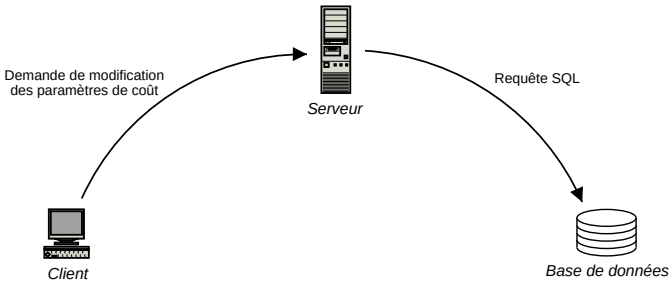


FIG. 4.10: Principe de la sauvegarde des paramètres de coût dans la base de données

La sauvegarde des paramètres spécifiques à chacune des machines est plus difficile à mettre en œuvre. Elle nécessite en effet de pouvoir choisir une machine avant de demander la valeur de C^P_{coeff} pour celle-ci.

Ainsi, nous mettons en place l'interface graphique représentée par la figure 4.11.

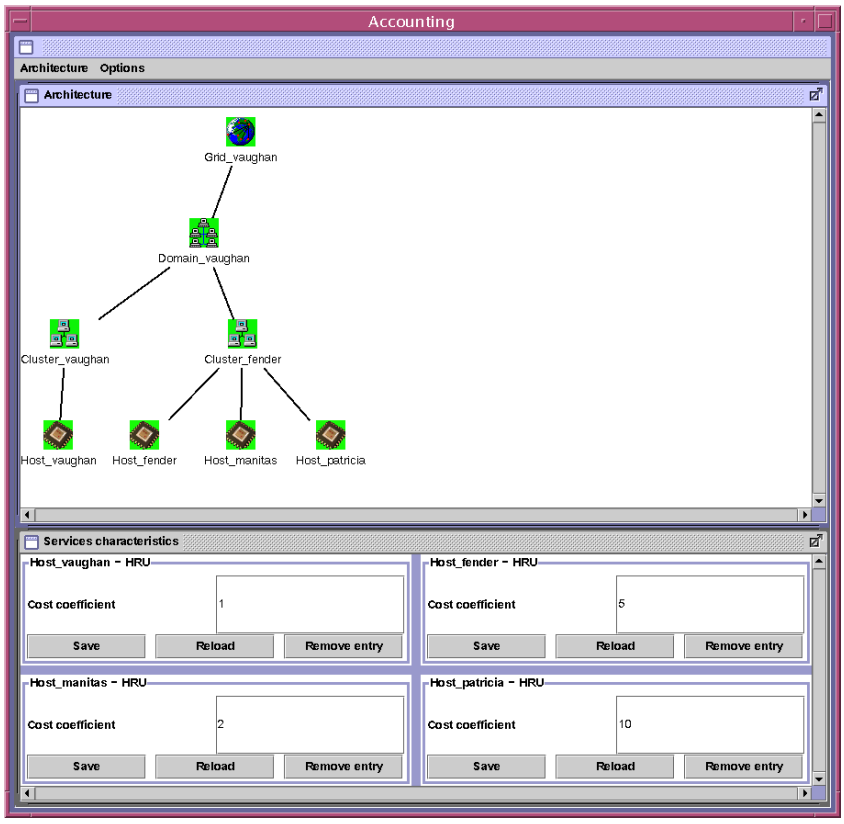


FIG. 4.11: Fenêtre permettant d'ajuster le coût de chacune des machines

Cette fenêtre est décomposée en deux parties. Une première fenêtre interne contient un arbre représentant tous les serveurs que contrôle Aroma. Ces serveurs sont situés aux différents niveaux de l'architecture d'une grille (serveurs de niveau grille, de niveau domaine, de niveau *cluster* et enfin de niveau hôte). Les serveurs

de niveau hôte sont ceux de plus bas niveau. Ils sont lancés sur toutes les machines utilisées par Aroma pour exécuter des tâches. Ainsi, le paramètre C^P_{coeff} doit être ajusté pour chacune de ces machines.

L'arbre présent sur l'interface graphique est construit dynamiquement à partir d'un fichier de configuration comportant la description de l'architecture des serveurs d'Aroma. En cliquant sur un serveur de niveau hôte, l'utilisateur a accès à la valeur de C^P_{coeff} correspondante, cette dernière se plaçant dans la seconde fenêtre interne de l'interface.

Tout comme pour les paramètres communs à toutes les machines, cette interface récupère les nouvelles valeurs fournies par l'utilisateur, les communique au serveur de plus haut niveau de l'architecture, qui se charge alors de les stocker dans la base de données (figure 4.10).

Une table de la base de données est dédiée au stockage de ces valeurs. Elle possède trois champs : le nom de la machine, la valeur de C^P_{coeff} qui lui est associée, et également la date de modification afin de pouvoir conserver un historique des différentes valeurs de ce paramètre pour chaque machine au cours du temps.

4.3.2 Prise en compte des mécanismes de facturation au niveau de l'ordonnanceur

Les paramètres de coût étant accessibles à partir de la base de données, nous pouvons maintenant modifier l'ordonnanceur d'Aroma afin qu'il prenne en compte les aspects liés à la facturation.

Avant de poursuivre, nous allons donner quelques précisions à propos de cet ordonnanceur. Son rôle est simple. Dès qu'il reçoit une requête de placement pour une application, il se charge de répartir les tâches qui la composent sur les différentes machines correspondant aux serveurs de niveau hôte d'Aroma. Les tâches sont alors exécutées sur ces machines. Notons que l'ordonnancement est choisi en vue d'assurer la qualité de service demandée pour chaque application lancée.

Côté client, une interface permet de se connecter aux serveurs d'Aroma pour leur confier l'exécution d'applications. Grâce à cette interface, l'utilisateur peut fixer tous les paramètres de l'application (nom de l'exécutable, liste d'arguments, nombre de tâches, temps processeur demandé, etc.).

La figure 4.12 montre les communications qui sont mises en place entre les différents acteurs lors du placement d'une application.

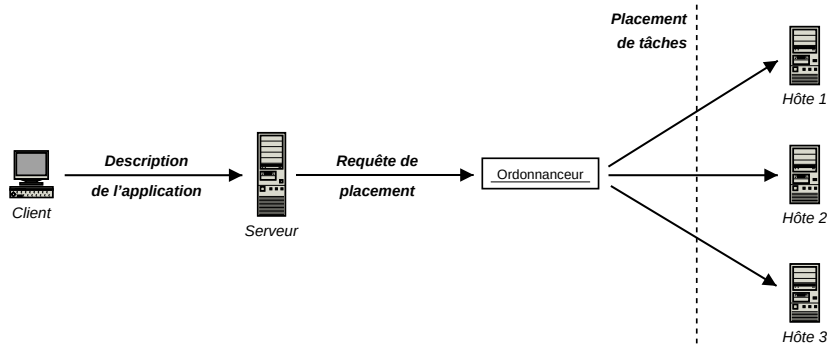


FIG. 4.12: Communications échangées lors du placement d'une application

Notre objectif est donc de modifier l'ordonnanceur afin de pouvoir récupérer les ressources consommées par une application, les facturer, et communiquer un prix au client.

Nous ne nous concentrerons pas ici sur les aspects techniques de cette opération. Les modifications du code source de l'ordonnanceur ont été effectuées dans le but de récupérer les informations nécessaires à la facturation, c'est-à-dire :

- le nombre de machines utilisées,
- la puissance de ces machines.

Toutes les autres informations seront fournies par l'utilisateur lorsqu'il donnera les caractéristiques de l'application à exécuter (figure 4.13). Quant aux paramètres de coût, ils seront extraits de la base de données.

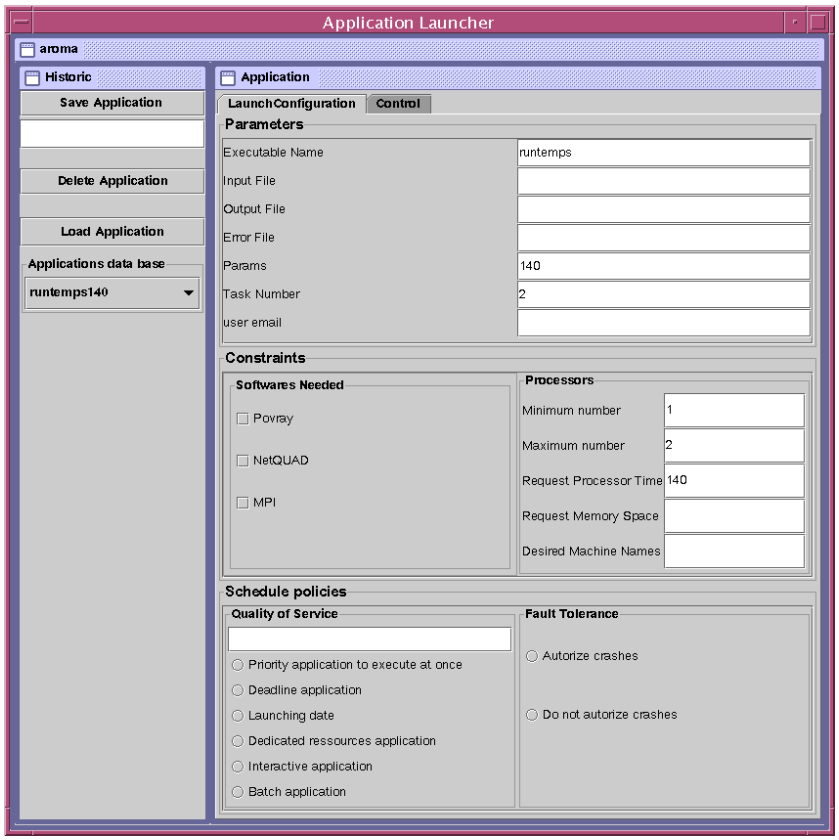


FIG. 4.13: Interface permettant à l'utilisateur de renseigner les paramètres décrivant l'application

Ainsi, nous allons consacrer cette partie à l'étude des résultats fournis par l'ordonnanceur que nous avons modifié. L'objectif est de démontrer que ces résultats sont conformes à nos attentes.

Il est à noter que, désormais, l'ordonnanceur calcule un prix, et que, lorsque l'application est terminée, celui-ci est transmis au client. Nous avons donc modifié l'interface graphique afin qu'il puisse être affiché à l'écran (figure 4.14).

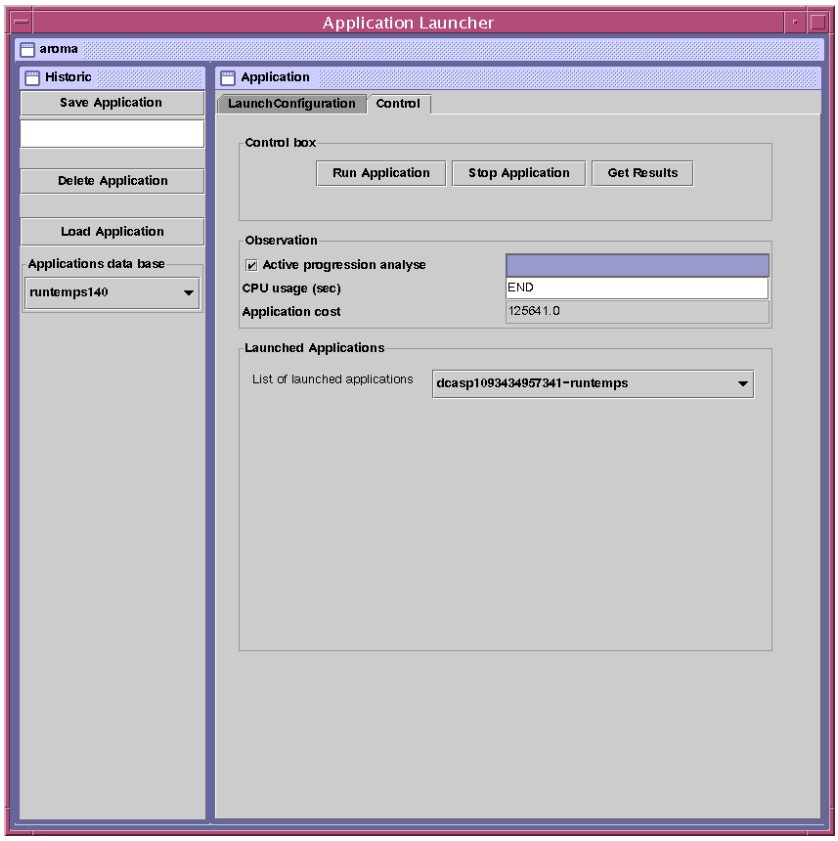


FIG. 4.14: Interface permettant à l'utilisateur de suivre l'évolution de l'application

Afin de prouver le bon fonctionnement des mécanismes de facturation mis en place, nous allons mener des tests sur quatre machines ($N^P = 4$). Ces machines possèdent toutes la même puissance, qui correspond à la puissance de référence :

$$\forall j \in \{1, ..., N^P\} \qquad P(j) = P^R = 502$$

Chaque machine aura son propre coefficient permettant d'ajuster localement le coût lié à la puissance de son processeur. Le tableau suivant donne la valeur de ce coefficient pour chacune des machines :

Nom de la machine	C^P_{coeff}
vaughan	1
manitas	2
fender	5
patricia	10

Nous pouvons enfin noter la valeur que nous avons choisie pour les paramètres de coût communs à toutes les machines :

$$C^T = 1 \qquad C^N = 1 \qquad C^P = 50$$

Nous lançons alors plusieurs applications, avec divers paramètres, et nous relevons le coût de chacune d'entre elles. Les résultats de ces tests sont donnés dans le tableau suivant :

Numéro du test	*Nombre total de tâches*	*Durée d'une tâche*	*Hôtes utilisés*	*Nombre de tâches sur chaque hôte*	*Coût de l'application*
1	1	140	vaughan	1	25241
2	1	140	fender	1	125641
3	1	20	fender	1	125521
4	2	140	fender patricia	1 1	376782
5	5	140	fender patricia manitas vaughan	2 1 1 1	578004

En comparant les deux premiers tests, nous observons que le coefficient appliqué localement à chaque machine est bien pris en compte. En effet, ce coefficient vaut 1 pour *vaughan* et 5 pour *fender*. Ce rapport se retrouve au niveau du prix.

Les tests 2 et 3 nous montrent que, en utilisant la même machine, et en faisant varier le temps d'exécution de chaque tâche, le coût ne varie que très sensiblement. Nous pouvons en conclure deux choses : d'une part, le temps d'exécution est bien facturé (sinon, le coût ne varierait pas du tout), et d'autre part, son importance dans le prix à payer est faible (ce qui est prévisible, puisque le coefficient lié au temps d'exécution est très petit devant celui lié à la puissance des processeurs).

Enfin, les derniers tests nous montrent que lorsque plusieurs machines sont utilisées, chacune d'entre elles est bien facturée au client.

4.3.3 Conclusion

Dans cette section, nous avons modifié l'ordonnanceur d'Aroma afin qu'il prenne en compte les aspects liés à la facturation. Les tests que nous avons ensuite effectués nous permettent de conclure que cet ordonnanceur facture le client conformément au modèle économique que nous avons établi.

Cependant, cet ordonnanceur ne cherche absolument pas à minimiser le coût d'une application. L'algorithme utilisé permet simplement de garantir à l'utilisateur une certaine qualité de service. Si plusieurs ordonnancements satisfaisant cette qualité de service sont possibles, l'algorithme ne choisit pas forcément celui qui a le coût le plus faible.

Par conséquent, il sera nécessaire, par la suite, de créer un nouvel ordonnanceur qui aura pour rôle de respecter le modèle économique que nous avons créé. Ainsi, il devra être capable de trouver un ordonnancement permettant de minimiser un compromis entre le coût d'une application et sa durée d'exécution, tout en respectant une qualité de service donnée.

Nous pouvons également noter que nous avons seulement inclus, dans les ressources à facturer, celles liées aux processeurs. Un modèle économique plus complet, prenant en compte toutes les ressources pouvant être facturées telles que la mémoire ou bien les librairies utilisées (cf. section 3.4.1), pourra être développé et inclus dans la future version de l'ordonnanceur.

Conclusion

Depuis quelques années, l'augmentation de la puissance des stations de travail semble ne connaître aucune limite. Cependant, de nombreuses applications scientifiques ou industrielles ont des besoins de plus en plus importants en termes de puissance de calcul, si bien que, même si la puissance des stations de travail augmente, une seule d'entre elles ne suffit pas pour exécuter ces applications. Le *grid computing* permet de répondre à ce problème en répartissant la charge de travail sur plusieurs machines.

Ainsi, mon stage m'a permis de découvrir les dernières technologies concernant ce domaine. J'ai pu examiner le fonctionnement général d'un intergiciel tel que Globus, et j'ai également appris que les modèles économiques du marché réel peuvent s'appliquer au domaine du calcul sur grille, afin de permettre une gestion des ressources basée sur les attentes et objectifs des propriétaires et des utilisateurs de ces ressources.

L'objectif principal du stage a été atteint. J'ai pu mettre en place un modèle économique, et aboutir à la conclusion qu'une approche basée sur ces modèles permet d'obtenir de bons résultats au niveau de l'ordonnancement de tâches sur les différentes machines d'une grille.

Grâce à ce stage, j'ai pu mieux cerner les travaux qui peuvent être menés dans un laboratoire de recherche. J'ai eu l'occasion d'aborder mon travail d'une autre façon que celle employée pour mener à bien mon projet de fin d'études.

Si, sur le plan technique, ce stage a été une réussite totale, il l'a été tout autant sur le plan humain. En effet, j'ai passé ces quelques mois dans un groupe de recherche qui m'a accueilli très chaleureusement. Les chercheurs et les doctorants se sont toujours montrés très disponibles afin que mon stage se déroule dans les meilleures conditions possibles.

Les recherches que j'ai effectuées sur les modèles économiques m'ont permis d'aboutir à la création d'un modèle économique simple pour Aroma. J'aurai désormais l'occasion d'étendre ces travaux dans le cadre de la thèse que je m'apprête à préparer. Je pourrai ainsi créer un modèle économique plus complet, prenant en compte d'autres ressources, et mettre en place un ordonnanceur basé sur ce modèle.

Bibliographie

[Berstis, 2002] Berstis V. *Fundamentals of Grid Computing.* IBM Redpaper, 2002.
Réf : `http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf`

[Buyya, 2002] Buyya R. *Economic-based Distributed Resource Management and Scheduling for Grid Computing.* PhD Thesis, Monash University, 2002.
Réf : `http://www.buyya.com/thesis/`

[Chetty & Buyya, 2002] Chetty M. & Buyya R. *Weaving Computational Grids : How Analogous Are They With Electrical Grids ?* Computing in Science and Engineering, 2002.
Réf : `http://www.buyya.com/papers/WeavingGrid.pdf`

[Ferreira & al.] Ferreira L., Berstis V., Armstrong J., Kendzierski M., Neukoetter A., Takagi M., Bing-Wo R., Amir A., Murakawa R., Hernandez O., Magowan J. & Bieberstein N. *Introduction to Grid Computing with Globus.* IBM Redbook, 2002.
Réf : `http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf`

[Foster, 2002] Foster I. *What is the Grid ? A Three Point Checklist.* GRIDToday, 2002.
Réf : `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`

[Foster & al, 2002] Foster I., Kesselman C., Nick J. & Tuecke S. *The Physiology of the Grid : An Open Grid Services Architecture for Distributed Systems Integration.* Open Grid Service Infrastructure WG, Global Grid Forum, 2002.
Réf : `http://www.globus.org/research/papers/ogsa.pdf`

[Foster & Kesselman, 1997] Foster I. & Kesselman C. *Globus : A Metacomputing Infrastructure Toolkit.* Intl J. Supercomputer Applications, 1997.
Réf : `ftp://ftp.globus.org/pub/globus/papers/globus.pdf`

[Foster & Kesselman, 1998] Foster I. & Kesselman C. *The Globus Project : A Status Report.* Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, 1998
Réf : `ftp://ftp.globus.org/pub/globus/papers/globus-hcw98.pdf`

[Foster & Kesselman, 1999] Foster I. & Kesselman C. *The Grid : Blueprint for a Future Computing Infrastructure.* 1999.

[Gayola & al., 2002] Gayola G., Lecointe B., Bacquet P, Parot J.M., Garcia J.M., Monteil T., Pascal P & Richard S. *CASP : Clusters for Application Service Provider.* Journées 2002 Réseau National de Recherche et d'Innovation en Technologies Logicielles, Ateliers RNTL-ASTI Partenariat et Transfert de Technologie, Toulouse (France), 2002.

[Lock, 2002] Lock R. *An Introduction to the Globus Toolkit.* 2002.
Réf : `http://www.comp.lancs.ac.uk/computing/research/cseg/projects/dirc/papers/An\%20introduction\%20to\%20the\%20Globus\%20toolkit.doc`

[Pascal & al., 2002] Pascal P., Richard S. & Monteil T. *Architecture of a Grid Resource Manager.* PDPTA'02, Las Vegas (USA), 2002.

[Quinson, 2003] Quinson M. *Découverte automatique des caractéristiques et capacités d'une plate-forme de calcul distribué.* Thèse à l'ENS de Lyon, 2003.
Réf : `http://tel.ccsd.cnrs.fr/documents/archives0/00/00/61/69/tel-00006169-00/tel-00006169.pdf`

[Sandholm & Gawor, 2003] Sandholm T. & Gawor J. *Globus Toolkit 3 Core - A Grid Service Container Framework.* 2003.
Réf : `http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf`

[Sotomayor, 2004] Sotomayor B. *The Globus Toolkit 3 Programmer's Tutorial.* 2004.
Réf : `http://www.casa-sotomayor.net/gt3-tutorial/index.html`

[Tuecke & al, 2003] Tuecke S., Czajkowski K., Foster I., Frey J., Graham S., Kesselman C., Maguire T., Sandholm T., Vanderbilt P. & Snelling D. *Open Grid Services Infrastructure (OGSI) Version 1.0.* Global Grid Forum Draft Recommendation, 2003.
Réf : `http://www.globus.org/research/papers/Final_OGSI_Specification_V1.0.pdf`

[Wang & Browne, 2003] Wang L. & Browne J. *Writing Grid Service Using GT3 Core* 2003.
Réf : `http://www.cs.utexas.edu/users/browne/cs395f2003/projects/WangReport.pdf`