Simulation et conception de services déportés sur grappes

Samuel Richard

THÈSE

Préparée au

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

En vue de l'obtention du

Doctorat de l'Institut National des Sciences Appliquées de Toulouse
Spécialité Systèmes Informatiques

par

Samuel Richard

Titre de la thèse :

Simulation et conception de services déportés sur grappes.

Soutenue le 9 Juin 2006 devant le jury :

M. :	Germain	GARCIA	Président
MM. :	Van-Dat	Cung	Rapporteurs
	Michel	Daydé	
MM. :	Bertrand	PEREZ	Examinateurs
MM. :	Jean-Marie	GARCIA	Directeurs de thèse
	Thierry	MONTEIL	

Remerciements.

Les travaux présentés dans ce manuscrit sont l'aboutissement de trois années et demi d'études réalisées au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS) dans le groupe Réseaux et Systèmes de Télécommunications (RST).

Je tiens à exprimer toute ma reconnaissance à Messieurs Jean-Claude Laprie et Malik Ghallab, successivement directeurs du laboratoire, pour m'avoir accueilli au LAAS et avoir mis à ma disposition les moyens nécessaires à la réalisation de mes travaux.

Je remercie Messieurs Jean-Marie Garcia et Thierry Monteil pour m'avoir offert la possibilité de découvrir le monde de la recherche et avoir encadré mes travaux.

Je suis très reconnaissant envers Messieurs Van-Dat Cung et Michel Daydé d'avoir accepté d'être les rapporteurs de cette thèse. Je les remercie tout particulièrement.

Je remercie Messieurs Germain Garcia, et Bertrand Perez d'avoir accepter de faire partie de mon jury de thèse.

Je tiens à remercier tout particulièrement Olivier Brun pour sa compétence son avis toujours éclairé et David Gauchard pour ses compétences techniques et sa disponibilité. Leur aide m'a été d'un grand soutien.

Je n'oublie pas les bons moments passés au sein du groupe RST et en remercie tous les membres. Je remercie tout particulièrement les thésards et les stagiaires pour leur bonne humeur : Bernard, Cédric, Charles, Cyril, Eric, Erika, Franck, François, Hassan, Mickael, Patricia, Philippe...

Merci aussi à tous les amis qui m'ont soutenu durant ces années, soit directement au LAAS (Nicolas, Vincent...) soit à l'extérieur du laboratoire : Magali, Sophie, Franck, Nicolas, Cyril, Hervé, Christine, Estelle, Stéphane, Mathieu...

Enfin, mes derniers remercieront vont à mes proches qui m'ont toujours accompagné et soutenu dans les bons moments, comme dans les périodes difficiles.

Table des matières

Table des figures

Chapitre 1

Introduction.

La démocratisation récente de l'Internet, et plus généralement des réseaux de communication haut débit à moindre coût, permet le développement de nouvelles formes d'utilisation de l'informatique.

L'utilisation de l'outils informatique, sous forme de services déportés, se démocratise peu à peu, et un panel de plus en plus diversifié de services connaissent un succès grandissant.

Le cluster ou grappe de machine est, à l'heure actuelle, le support le plus apte à rendre ce service grâce à ses qualités en terme d'extensibilité, de modularité, d'évolutivité et de coût. Néanmoins, la mise en place de ce type de support d'exécution fait apparaître de nouvelles problématiques, ou nécessite l'adaptation de problématique existantes.

L'objectif de cette thèse est d'étudier les problématiques liées à la fois à la conception d'intergiciels [1] permettant la mise en place de services déportées sur des grappes de machines, et d'étudier les performances de ce type d'architecture.

1.1 La gestion des ressources

L'utilisation de grappes de machines a tout d'abord vu le jour dans le domaine du calcul haute performance. Les machines parallèles, aux architectures propriétaires, cèdent peu à peu leur place à des grappes et grilles de calcul. L'intérêt de ce support réside principalement dans son rapport prix/performance. Limitées dans un premier temps à quelques machines homogènes, les architectures actuelles évoluent vers la mise en commun d'un nombre plus important de machines, pouvant avoir des architectures hétérogènes. Ce type d'architecture cumule des avantages en termes de coût d'acquisition (utilisation ou réutilisation d'architectures matérielles grand public) et d'évolutivité (le nombre de machines d'une grappe peut être étendu de manière illimitée).

La mise en place de services déportés sur des grappes de machines nécessite la mise en place d'intergiciels spécifiques afin de gérer les ressources pour assurer une haute disponibilité et une qualité de service apte à séduire les utilisateurs. L'utilisation de ressources distantes peut également donner lieu à une facturation du service. Il est dans ce cas judicieux de définir plusieurs classes d'utilisateurs pour différencier ceux qui sont prêts à payer plus pour un service

[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 grappe.

offrant un meilleur temps de réponse (par allocation d'un plus grand nombre de ressources ou par une priorité supérieure aux autres utilisateurs, par exemple).

Les problématiques adressées par ces intergiciels sont :

- l'authentification : seuls les utilisateurs autorisés peuvent avoir accès à la machine ; la classe de service maximale de l'utilisateur peut être déduite de l'identification de l'utilisateur,
- la confidentialité : les données des travaux d'un utilisateur ne doivent pas être observées lors de leur transit sur le réseau, ni lors de leur exécution,
- la haute disponibilité pour assurer la fiabilité des opérations et des résultats obtenus,
- la gestion des ressources : cette problématique concerne à la fois l'observation des ressources et la sélection des ressources utilisées au moyen de politiques d'ordonnancement spécifiques.

1.2 Les applications concernées

De nombreuses applications sont concernées par l'utilisation en mode déportée. Dans le domaine industriel, de nombreuses applications portées sur des grappes de machines peuvent utiliser ce modèle d'exécution dans des domaines aussi variés que la physique nucléaire, la bio-informatique, la santé, l'écologie, les science de l'univers [49] ou les télécommunications. Ces outils :

- sont souvent de gros logiciels,
- sont souvent onéreux,
- nécessitent un ensemble de bibliothèques annexes (Plugins),
- nécessitent une assistance suivant les périodes, suivant les problèmes traités,
- nécessitent parfois des machines puissantes (en processeur, en mémoire ou en capacité de stockage),
- l'utilisation peut être saisonnière ou ponctuelle et donc coûteuse en terme d'achat de licence annuelle.

Suivant les cas, et pour différentes raisons (financières, ressources humaines disponibles, complexité de mise en place) l'utilisateur potentiel de tels produits, peut être en difficulté pour une utilisation efficace de ces applicatifs sur la machine de son bureau. D'autre part, il doit très souvent faire face à des problèmes contraints par un délai de réaction très court. Dans ce cas, l'utilisation de moyens et de services déportés au sens large, vont permettre à cet utilisateur de résoudre son problème à moindre coût et dans un temps minimal.

Les applications de calcul scientifiques ne sont pas les seules à pouvoir tirer profit de l'utilisation de grappes de machines. Que ce soit dans le cadre des services publics (déclaration d'impôts, téléchargement de formulaires administratifs...), du service bancaire (gestion de compte, achats en bourse...), de l'assurance (déclaration de sinistres, réalisation de devis...), du divertissement (achat de musique en ligne, logos de téléphones portables, vidéo à la demande...), de nombreux services, toujours plus complexes, connaissent un succès grandissant. Ces applications rencontrent les mêmes besoins en terme de haute disponibilité, de gestion de ressources et parfois d'authentification que les applications scientifiques.

De ce fait, une même grappe de machines peut à la fois héberger des services multimédia (serveur Web de l'entreprise, service de télé-formation...) et des applications scientifiques utilisées en mode déporté. Les performances de ces différentes classes d'application doivent

être prises en compte afin d'être capable de dimensionner les grappes de machines à utiliser pour offrir un niveau de qualité de services en adéquation avec les attentes des utilisateurs.

1.3 Plan de la thèse

L'objectif de cette thèse est de proposer un intergiciel permettant l'exécution déportée d'applications scientifiques sur des grappes de machines, et d'étudier les performances de ce système afin de pouvoir le dimensionner.

Le premier chapitre positionne la problématique de la thèse au sein de la nébuleuse des grappes et des grilles informatiques. Ces deux concepts y sont définis et les différentes vision de la grille informatique y sont présentées.

Le deuxième chapitre présente les différents outils existants adressant les problématiques liées à l'exécution distante d'applications scientifiques sur des grappes de machines. Les outils d'observation des ressources, et différents intergiciels pour les grilles informatiques y sont étudiés.

Les deux chapitres suivants détaillent le fonctionnement du gestionnaire de ressource Aroma (scAlable Resource Observer and wAtcher) réalisé durant cette thèse. Les problématiques d'observation des ressources, de haute disponibilité et les spécificités de l'utilisation déportée y sont notamment explicitées.

Le cinquième chapitre traite du dimensionnement de grappes de machines devant héberger à la fois des applications de calcul scientifique et des contenus interactifs. Le fonctionnement des différents composants ayant une influence sur les performances d'une grappe de machines y est étudié, et les modèles conçus dans le but de simuler le fonctionnement d'un tel système y sont présentés.

Enfin, le dernier chapitre présente les résultats obtenus lors de la simulation du fonctionnement de grappes de machines. Ces résultats sont comparés à des modèle analytiques simples dans le cas d'applications de présentation de contenus multimédia, et à des mesures d'exécutions réelles dans le cas d'applications de calcul scientifique.

Chapitre 2

Les concepts de grappes et de grilles.

Sommaire

2.1 Introduction

L'augmentation constante des besoins en termes de puissance de calcul informatique a toujours été un défi auquel la communauté scientifique 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 permettent d'aboutir à la création de machines de plus en plus puissantes, une seule machine ne fournit toujours pas suffisamment de puissance pour effectuer des calculs d'une complexité trop élevée, ou traitant un trop grand nombre de données.

Jusqu'à la dernière décennie, la seule solution viable pour résoudre ces problèmes complexes étaient les machines parallèles. Ces dernières permettent d'intégrer dans un même bâti plusieurs processeurs reliés entre eux par des bus de communication très rapides et disposant d'une grande quantité de mémoire. Bien que très efficaces, ces machines présentent l'inconvénient d'être des solutions spécifiques, propriétaires et s'avèrent de ce fait très coûteuses et peu évolutives. Elles obligent également le portage des codes de calcul et souvent leur réécriture afin de bénéficier au mieux des performances offertes par chaque architecture.

La démocratisation des ordinateurs personnels, l'augmentation constante de leurs performances et la démocratisation des réseaux de communication haut débit, offrent aujourd'hui des alternatives sérieuses à ces solutions onéreuses. Ces systèmes appelés grappes et grilles

de calculs reposent sur la mise en commun d'un nombre important de machines "standard" reliées entre elles par des réseaux haut débit.

Bien qu'apportant une réponse aux problèmes de coût, d'évolutivité et de portage du code, ces solutions soulèvent d'autres problèmes en termes de mise à l'échelle, de fiabilité, d'hétérogénéité, de sécurité et de gestion des ressources.

Dans un premier temps, une définition des concepts de grappes et de grilles sera donnée. Les objectifs des grilles de calcul ainsi que les problèmes soulevés par la gestion de ressources dans de tels environnements seront ensuite introduits. Finalement, les principaux domaines d'application des grilles de calcul seront présentés.

2.2 Grappes et Grilles

2.2.1 Définitions

Une grappe de machine (Cluster en Anglais) désigne un ensemble d'ordinateurs, appelés nœuds, tous inter-connectés, dans le but de partager des ressources informatiques. Une grappe peut être constituée d'ordinateurs de bureaux (figure 2.1(a)), de "racks" de machines constituées de composants standards (figure 2.1(b)) ou de "lames" (figure 2.1(c)) également constituées de composants standards afin d'optimiser l'espace physique. Une grappe est généralement composée de machines homogènes en termes d'architecture et de système d'exploitation. Elle ne regroupe que des machines appartenant au même domaine d'administration réseau et les nœuds communiquent entre eux en utilisant un réseau de communication rapide (100Mb ou Gb). Les différents nœuds d'une grappe possèdent souvent une configuration logicielle semblable. Une pratique courante, utilisée par la plupart des logiciels d'administration de grappes (Rocks Cluster Distribution[63], IBM CSM[30], SUN Cluster[73]) est d'installer les logiciels sur un nœud maître, puis de déployer une image du système sur chaque nœud de la grappe. Ce procédé permet de maintenir une certaine cohérence entre les différents nœuds tout en limitant les opérations d'installation au seul nœud maître.

(a) Grappe de PC (b) Rack de machines (c) Lames de machines

Fig. 2.1 – Les différentes grappes

Le terme grille quant à lui, désigne un ensemble beaucoup plus important de machines, hétérogènes, réparties sur différents domaines d'administration réseau. Les différentes entités composant une grille peuvent être réparties sur l'ensemble de la planète et communiquent

entre elles en utilisant une grande diversité de réseaux allant de l'Internet à des réseaux privés très haut débit.

Le concept de grille informatique puise son inspiration dans le développement des grilles d'électricité (*power grids*) au début du XXème siècle [46]. 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 [28]. Les composants formant le réseau électrique sont hétérogènes, et la complexité induite est totalement masquée à l'utilisateur final. Ainsi, une grille informatique possède les mêmes propriétés d'hétérogénéité des ressources que le réseau électrique, le défi scientifique est d'offrir à l'utilisateur de la grille la même transparence d'utilisation qu'offre la prise électrique à l'utilisateur du réseau électrique.

Une grille informatique est une infrastructure matérielle et logicielle qui fournit un accès consistant et peu onéreux à des ressources informatiques [28]. Le but est ainsi de fédérer des ressources provenant de diverses organisations désirant collaborer en vue de faire bénéficier aux utilisateurs d'une capacité de calcul et de stockage qu'une seule machine ne peut fournir. Cependant, tout système informatique distribué ne peut posséder l'appellation de grille de calcul [25]. En effet, une grille est un système qui coordonne des ressources non soumises à un contrôle centralisé, qui utilise des protocoles et interfaces standards dans le but de délivrer une certaine qualité de service (en termes de temps de réponse ou bien de fiabilité par exemple).

Plusieurs types de grilles peuvent être discernées selon le type de ressources utilisées et l'utilisation recherchée. La section suivante propose une classification des différents types de grilles.

2.2.2 Différentes visions d'une Grille

Classification par type de ressource partagée

Les différents projets de recherche ayant vu le jour à l'heure actuelle autour des grilles informatiques permettent de mettre en évidence une première classification des grilles par type de ressources partagées :

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

Différentes visions des grilles de calcul

Les grilles de calcul peuvent elles-mêmes être classifiées en sous-catégories selon l'utilisation recherchée :

- *Virtual Supercomputing :* ce terme désigne une association de plusieurs supercalculateurs géographiquement répartis. Chaque nœud est une machine parallèle contrôlée par un gestionnaire de tâches réalisant du placement par lots. Ce type d'environnements est

destiné à des applications gros grain, ou à des applications offrant plusieurs niveaux de parallélisme : un premier niveau exploitable par une machine parallèle et un deuxième exploitable par un système distribué.
- *Internet computing :* cette vision des grilles de calcul se caractérise par la mise en commun d'un très grand nombre d'ordinateurs personnels. Le but recherché est d'utiliser les périodes durant lesquelles l'ordinateur est inutilisé. Ce type d'utilisation est destiné à des applications au grain fin pour lesquelles un traitement peut être facilement interrompu ou re-exécuté. Il permet toutefois de disposer d'une immense puissance de calcul potentielle à moindre coût.
- *Metacomputing :* ce terme désigne la mise en commun de plusieurs machines ou groupes de machines, chaque machine donnant accès à un logiciel particulier. L'intérêt de ce type de grilles est de partager des coût importants de logiciels entre différentes organisations ou de permettre l'accès à des logiciels nécessitant du matériel spécifique à moindre coût.

2.3 Les grilles de calcul

2.3.1 Objectifs du "grid computing"

Le calcul sur grille possède plusieurs objectifs [79] :

- **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 de ressources, tels que des équipements spéciaux, des logiciels, et bien d'autres services. Certaines machines peuvent, par exemple, héberger des logiciels ayant des coûts de licence très élevés. Ainsi, l'utilisation d'une grille de calcul permet de bénéficier de ces logiciels à des coûts moindres. De la même manière, 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 moins sollicitées.

- **Gérer des applications avec *deadline* proche :** Si une application doit être exécutée avec une contrainte de date butoir très proche, l'utilisation d'une grille peut s'avérer utile. En effet, si l'application peut être découpée en un nombre suffisant de tâches, et si une quantité adéquate de ressources peut lui être dédiée, l'application bénéficiera d'une capacité de calcul suffisante pour être exécutée tout en respectant une *deadline* proche.

- **Assurer une tolérance aux fautes pour un coût moindre :** Dans les systèmes conventionnels, la tolérance aux fautes est réalisée grâce à la redondance du matériel sensible. Cette solution possède l'inconvénient d'avoir un coût assez élevé. Les grilles de calcul, de par leur nature, offrent une solution alternative pour effectuer de la tolérance aux fautes. En effet, si une défaillance apparaît 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 fournissent pour plus de fiabilité.

2.3.2 Problèmes inhérents à la gestion d'une grille de calcul

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. Les spécificités des grilles de calcul rendant leur gestion délicate [47] vont maintenant être présentées.

Hétérogénéité

La première caractéristique des ressources constituant une grille est sans nul doute leur hétérogénéité. Qu'elles soient matérielles ou bien logicielles, les ressources sont souvent très différentes les unes des autres. Cette hétérogénéité impose des contraintes de portage de code, d'utilisation de langages multi-plateformes et d'utilisation de protocoles de communication standardisés. L'utilisateur doit de plus pouvoir utiliser la grille de manières transparente et homogène quelle que soit l'architecture de sa propre machine et l'architecture de la machine à laquelle il se connecte.

Multiplicité des domaines d'administration

Les ressources sont géographiquement distribuées et appartiennent à différentes organisations indépendantes. Elles appartiennent donc à plusieurs domaines d'administration distincts, ayant chacun sa propre politique de gestion et de sécurité (figure 2.2) en terme d'accès au réseau, d'accès aux données, d'authentification ou encore de confidentialité. Ainsi, les personnes chargées d'administrer la grille n'auront pas forcément de privilèges particuliers sur les machines des différents domaines d'administration. Il est donc indispensable de mettre au point des méthodes d'administration particulières ne nécessitant aucun privilège sur les machines cibles. Il est également indispensable que chaque administrateur local conserve ses propres privilèges sur les machines qui lui appartiennent.

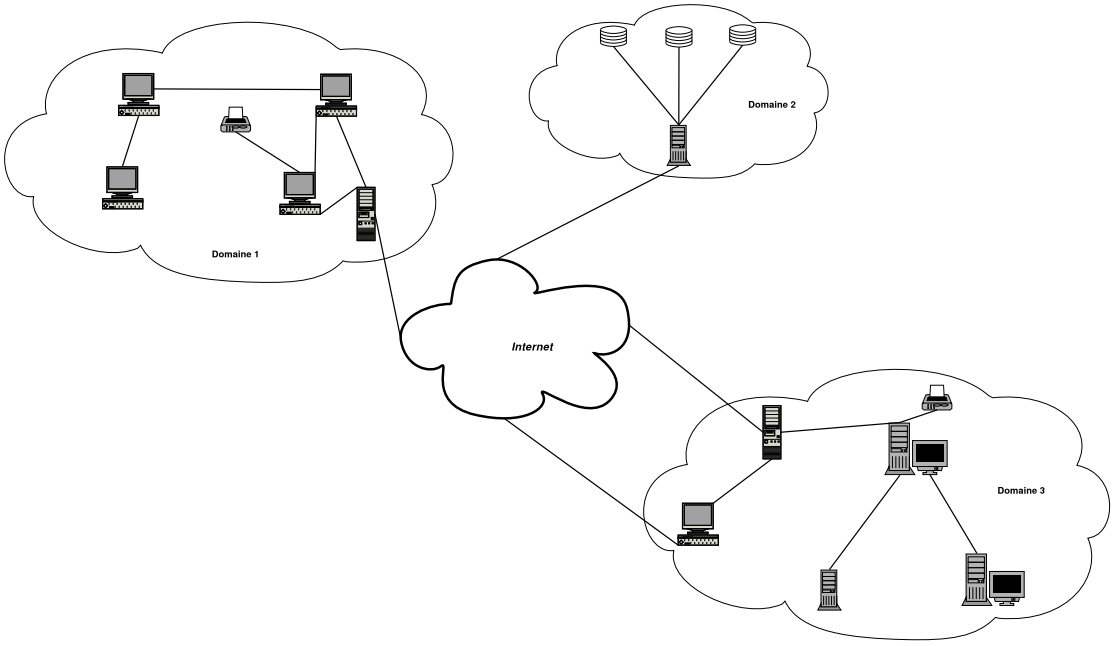


Fig. 2.2 – Répartition des ressources d'une grille entre différents domaines

Aspect dynamique

Du fait du grand nombre de ressources considérées, la défaillance d'une ressource (machine ou réseau) est un événement courant qui ne doit pas mettre en péril le fonctionnement de la grille. Le gestionnaire de ressources tout comme les applications doivent tenir compte de cet aspect et être capables de réagir rapidement à la perte ou à l'ajout d'une machine. De la même manière, l'ajout de fonctionnalités à un gestionnaire de ressources doit pouvoir se faire, dans la mesure du possible, sans qu'il soit nécessaire de réinstaller le gestionnaire sur l'ensemble des nœuds de la grille.

Gestion des ressources

La capacité de mise à l'échelle d'une grille est également une caractéristique à prendre en compte. En effet, une grille pourra être constituée d'une dizaine de ressources, tout comme de

plusieurs milliers de ressources. Ce problème de dimensionnement pose de nouvelles contraintes sur les applications et les algorithmes de gestion des ressources. L'observation des ressources notamment, ne doit pas induire une sur-consommation de ressources trop importante et ce quelque soit le nombre de nœuds de la grille.

2.3.3 Architecture type d'une grille de calcul

L'architecture d'une grille peut être vue de plusieurs façons. Nous choisirons ici de la représenter selon une architecture en couches [46] (figure 2.3).

Fig. 2.3 – Architecture en couches d'une grille

La couche de plus bas niveau correspond aux ressources elles-mêmes, gérées par des gestionnaires locaux, et inter-connectées 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.). Les ordonnanceurs présents à ce niveau peuvent être de deux types, soit l'ordonnanceur local d'une machine unique, soit un ordonnanceur d'une grappe de machine tel que Condor [45][77][78], LSF [84] ou bien PBS [4][23].

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 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 placer les tâches qui leurs sont confiées sur des machines appartenant à des domaines réseaux distincts.

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 bibliothèques 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.

2.4 Domaines d'application et exemples

Les grilles de calcul, même si elles constituent une nouvelle technologie en cours de développement, ont plusieurs exemples concrets d'utilisation à leur actif [43]. Ces exemples d'utilisation permettent de faire apparaître cinq grands types de domaines d'application pour lesquels les grilles de calcul peuvent être utilisées [28] :

- **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, le domaine de la conception aéronautique, de l'analyse de risques 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. La *ZetaGrid* est le premier exemple de ce type de grille. 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. Le projet *SETI@home* est également

un des projets de calcul haut débit les plus populaires. 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.

- **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. Les projet *CERN openlab* est le projet phare de ce type d'application, son but est de pouvoir traiter jusqu'à un Pétaoctet de données.

- **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.

2.5 Conclusion

Le concept de grille de machines est un concept naissant en pleine évolution. Le but initial des grilles est de permettre d'accéder aux ressources informatiques aussi simplement que l'on accède au réseau électrique. A partir de ce concept sont apparues une multitude d'utilisations permettant de résoudre différentes classes de problèmes : accès transparent à des masses importantes de données, meilleure utilisation des ressources existantes, accès transparent à une grande puissance de calcul, travail collaboratif...

Notre étude s'intéresse au cas particulier des grilles de calcul dont le but est fournir une puissance de calcul "infinie" afin de résoudre des problèmes de plus en plus complexes. Dans ce cas précis d'utilisation, les grilles de calcul remplacent peu à peu les machines massivement parallèles traditionnellement utilisées. Bien que permettant de gommer certaines limites des machines parallèles, ces nouveaux environnements soulèvent de nouveaux problèmes du fait de leur hétérogénéité, de leur dispersion géographique et de leur aspect dynamique.

Les principaux intergiciels existants permettant de gérer les ressources d'une grille de calcul vont maintenant être présentés.

Chapitre 3

Les outils existants.

Sommaire

3.1 Introduction

La gestion de grappes et de grilles de machines soulève plusieurs problèmes pouvant être solutionnés par différents types d'outils : observation de ressources, placement de tâche, communication entre machines, sécurité... De plus, comme nous l'avons montré dans la section précédente, il existe plusieurs visions des grilles et de ce fait, divers environnements de gestion de grilles ont vu le jour ces dernières années. Chaque environnement permet d'adresser les problèmes spécifiques liés à sa propre vision des grilles, en reposant soit sur des outils existants soit sur des solutions nouvelles. Le domaine des grilles étant un domaine jeune en pleine croissance, de nouvelle solutions ou de nouvelles versions d'outils existants voient régulièrement le jour et ces versions correspondent parfois à des changements significatifs d'architecture.

Pour toutes les raisons énoncées précédemment, il est assez difficile de répertorier l'ensemble des outils permettant de gérer des ressources sur des grappes ou grilles de machines et il est également difficile de classifier ces différents outils selon les fonctionnalités ou le type d'utilisation qu'ils proposent. Les sections suivantes ont pour but de présenter les principaux acteurs de la gestion de ressources pour grappes et grilles. Cette présentation s'articule autour de quatre type d'acteurs : les outils d'observation de ressources (machines et réseau), les

environnements de gestion de ressources distribuées, les gestionnaires de grilles et les environnements client-serveur sur grilles.

3.2 Outils d'observations

3.2.1 Ganglia

Ganglia [48] est un outils d'observation pour environnements de calcul haute performance tels que les grappes et les grilles de calcul. Il utilise une représentation hiérarchique des grappes en grille afin de favoriser le passage à l'échelle (figure 3.1).

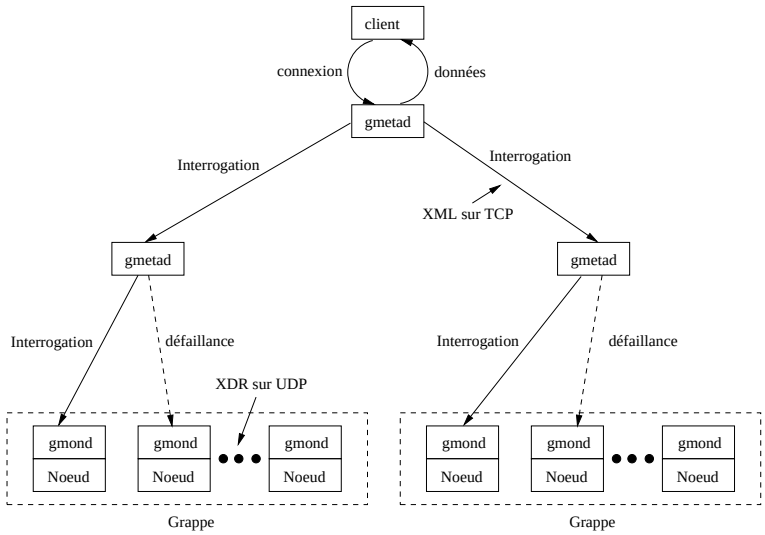


Fig. 3.1 – Architecture de ganglia

Ganglia repose sur l'utilisation de protocoles de communications multicasts et sur l'utilisation d'une structure hiérarchique de connexions point à point entre certain nœuds de la grappe afin de relier les grappes entre elles et d'agréger leurs informations d'état. Il utilise des technologies actuelles telles que XML pour la représentation de données, XDR pour la communication de données et RRDtool pour le stockage et la visualisation de données. L'implémentation de ganglia repose principalement sur deux *daemons* : gmond et gmetad.

gmond : les *daemons* gmond ont pour but de collecter les informations concernant une unique grappe. Un *daemon* est présent sur chaque nœud de la grappe, et communique avec les autres nœuds de la grappe en utilisant plusieurs protocoles multicasts. Ces communications multicasts permettent de : publier l'ajout d'un nouveau nœud, envoyer ses informations d'état aux autres nœuds de la grappe ou détecter la défaillance d'un nœud. Tous les nœuds de la grappe communiquent entre eux, de ce fait ils ont tous la même vision de l'état de la grappe.

gmetad : les *daemons* gmetad permettent de construire une représentation hiérarchique d'un ensemble de grappes. Chaque *daemon* collecte les informations concernant l'état de ses nœuds fils et en construit une représentation agrégée. Les nœuds fils peuvent être, soit les machines d'une grappe (au quel cas l'information agrégée est l'état de la grappe), soit d'autres

daemons gmetad (au quel cas l'information agrégée est l'état d'un ensemble de grappes). Les *daemons* gmetad communiquent entre eux en utilisant des communications point à point.

3.2.2 Network Weather Service

NWS est un outil d'observation fournissant la prédiction de performance de ressources dynamiques dans des environnements distribués. NWS prédit la performance réseau (latence et bande passante)[82][14] , le pourcentage de CPU disponible sur chaque machine qu'il contrôle [83] et la performance de la mémoire. NWS effectue des mesures périodiques sur la performance délivrable des ressources, utilise l'historique des mesures et des techniques statistiques pour la prédiction. Il communique ensuite les résultats de la prédiction aux schedulers. Il y a trois catégories de méthodes de prédiction : méthodes basées sur la moyenne utilisant une estimation de la moyenne comme prédiction, méthodes basées sur la médiane et les méthodes autoregressives. NWS choisit la meilleure prédiction pour une ressource en comparant les erreurs de prédiction avec les mesures. On distingue trois modules dans NWS : "sensory subsystem" qui collecte les informations sur la performance des ressources, "forecasting subsystem" qui prédit la performance et donne l'information à un "reporting subsystem". La figure 3.2 présente la structure de NWS.

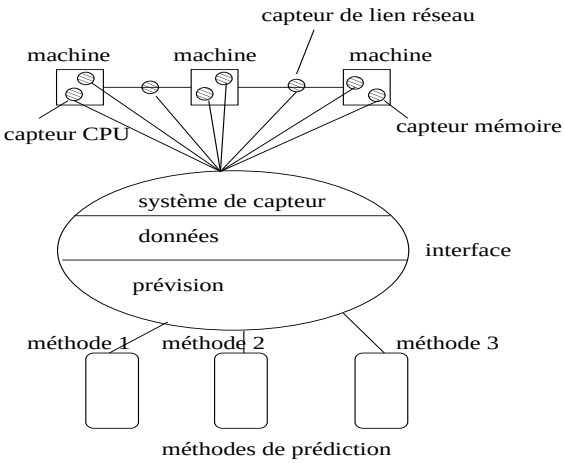


Fig. 3.2 – La structure du Network Weather Service

Un serveur NWS doit s'exécuter sur chaque machine qu'il supervise. Chaque serveur a un capteur de performance réseau et un capteur CPU. Tous les serveurs connaissent les machines supervisées et le numéro de port TCP auquel chaque serveur est relié.

3.3 Gestionnaires de Grilles

3.3.1 Globus

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. 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.

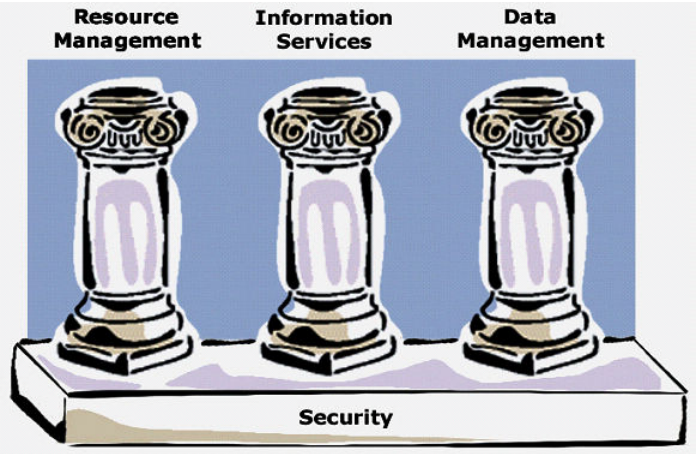
Les outils ainsi mis à disposition des utilisateurs ont pour but d'adresser plusieurs problèmes auxquels est souvent confronté le *grid computing*. Parmi ceux-ci figurent [26] :

- 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.

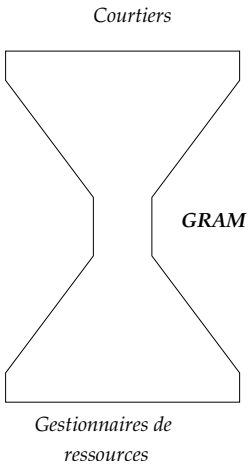
De manière générale, la composition de la *Globus Toolkit* peut être représentée par les trois piliers de la figure 3.3(a). Ainsi, trois composants essentiels de la *Globus Toolkit* sont à distinguer :

- 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.



(a) Les trois piliers de la Globus Toolkit



(b) Le GRAM au cœur d'une architecture en sablier

Fig. 3.3 – La Globus Toolkit

3.3.1.1 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 3.3(b) montre la place du GRAM au sein de l'architecture en couches de Globus [27]. 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.

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 pour 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 [62]. 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 récupérer les ressources manquantes. 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.

3.3.1.2 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 [27].

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 [62] (figure 3.4). Ces deux composants vont être successivement décrits.

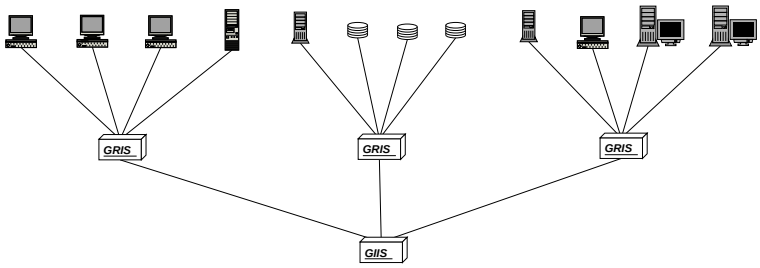


Fig. 3.4 – Architecture du service d'information

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.

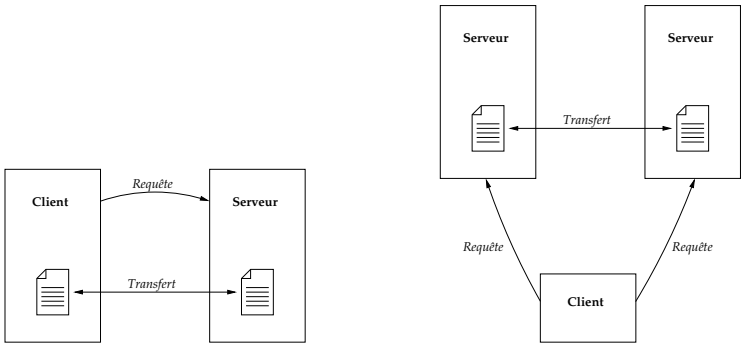
Il est à noter que la présence d'un seul serveur GIIS sur une grille constituerait un point fragile. A ce propos, des serveurs secondaires sont mis en place afin d'assurer une meilleure tolérance aux fautes.

3.3.1.3 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 [43]. 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 3.5) :

- 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. 3.5 – Modes de transfert du protocole GridFTP

3.3.1.4 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*).

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 pourra 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 mécanisme d'authentification mutuelle basé sur l'échange d'informations cryptées, est utilisé afin de garantir l'identité des entités communicantes.

Par la suite, 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.

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.

3.3.2 OGSA et les Grid Services

3.3.2.1 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 [29] 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.

En résumé, OGSA définit la manière dont 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.

3.3.2.2 Les Grid Services

Le terme de *Grid Service* désigne, de manière générale, un service disponible dans un environnement de grille [75]. 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*) [65], 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 s'exécutent 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.

Les différentes étapes réalisées lors de l'invocation d'un service à distance simple [7] sont les suivantes :

1. Le client lance une requête UDDI (*Universal Description, Discovery and Integration*) vers un *UDDI Registry* pour localiser les serveurs fournissant le service désiré.

[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.

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 sont résumées par le schéma de la figure 3.6.

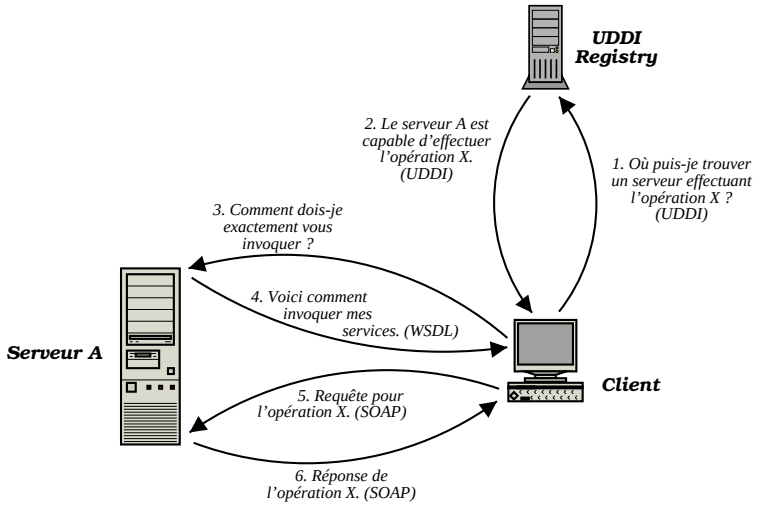


Fig. 3.6 – Principe de l'invocation d'un Web Service

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.

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.

3.3.3 Sun GridEngine

3.3.3.1 Fonctionnalités principales

Sun GridEngine [66], dans sa dernière version (*N1 Grid Engine 6*) [74] permet de gérer efficacement un ensemble de grappes de machines appartenant à une même organisation (concept appelé campus grids). *Sun GridEngine* permet la soumission de tâches interactives, de tâches

parallèles et de tâches en mode traitement par lot. Le système ordonnance les tâches soumises de manière à tenir compte des contraintes liées à chaque tâche (mémoire, date butoir, ...), des priorités données à chaque client et de la politique de gestion des ressources définie sur chaque site de la grille.

Chaque administrateur d'une grappe composant la grille peut définir la politique de gestion de ressources qu'il souhaite voir appliquer sur sa grappe. Quatre politiques sont définies par le système :

- *Urgency* : le système attribue une priorité de manière automatique à chaque tâche. La valeur de cette priorité est calculée en fonction des besoins en ressources de la tâche (bibliothèques spécifiques, quantité mémoire...), de la date butoir de la tâche et de la durée depuis laquelle la tâche est en attente d'exécution.
- *Functional* : les priorités entre les tâches sont définies en fonction de l'appartenance du client à un groupe d'utilisateurs ou à un projet spécifique.
- *Share-based* : chaque tâche a droit à un certain pourcentage de ressources. Des pourcentages peuvent être définis entre groupes d'utilisateurs, entre type d'applications...
- *Override* : ce mode permet à l'administrateur de la grappe d'intervenir manuellement sur la priorité de chaque tâche.

Lors de la soumission d'une tâche, un profil de cette tâche est constitué à partir de contraintes fournies par le client et de l'identité du client (appartenance à un groupe d'utilisateurs ou à un projet). Le système choisit ensuite la file d'exécution la plus adéquate pour cette tâche, en tenant compte du profil de la tâche, des politiques de gestion de ressources de chaque site et de l'état des ressources de chaque site. Une fois la tâche exécutée, une trace de son exécution est conservée.

3.3.3.2 Architecture

Grid Engine fait la distinction entre quatre types de nœuds (figure 3.7) : *master host, execution host, administrative host* et *submit host.*

- *Adminstrative Host* : ces nœuds permettent d'administrer la grille (définition de droits d'utilisateurs, de priorités...)
- *Submit host* : ces nœuds permettent à un utilisateur de soumettre et de contrôler l'exécution de tâches uniquement en mode traitement par lots. L'utilisateur se connecte sur un nœud *submit host* et peut soumettre directement une tâche en utilisant la commande *qsub* ou suivre l'état d'une tâche en cours de traitement grâce à la commande *qstat.*
- *Master host* : c'est le nœud central de l'activité d'une grappe. Ce nœud héberge deux types de daemons : *sge_ qmaster* et *sge_ sched* qui contrôlent l'ensemble des composants de la grille. Le daemon *sge_ qmaster* gère des informations d'état concernant les machines, les files d'exécution, les tâches, la charge du système et les permissions des utilisateurs. Il reçoit des décisions de placement du daemon *sge_ schedd* et transmet les requêtes de soumissions correspondantes aux daemons *sge_ execd* des nœuds concernés. Le daemon *sge_ schedd* utilise la connaissance que lui fournit le *sge_ qmaster* sur l'état d'une grappe afin de prendre des décisions d'ordonnancement. Deux types de décisions peuvent être prises : le placement d'une tâche sur un nœud ou la réorganisation et le changement de priorité de tâches déjà placées afin de satisfaire les contraintes de la nouvelle tâche.
  Par défaut, un nœud *Master host* joue également les rôles de *administrative host* et de *submit host.*

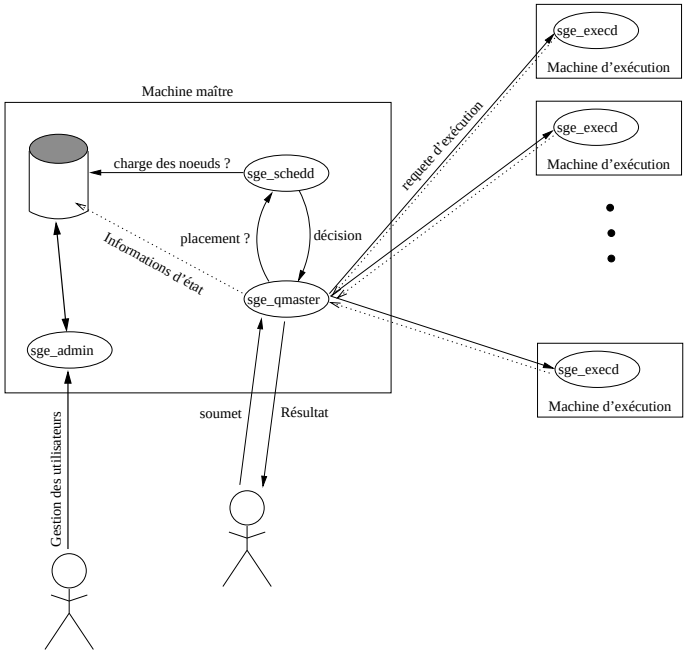


Fig. 3.7 – Composants de Sun Grid Engine et interaction avec un client

– *Execution Host* : ces nœuds sont chargés de l'exécution des tâches. Une file d'exécution est rattachée à chaque execution host. Un daemon *sge_ execd* est chargé de gérer sa file d'exécution, d'assurer l'exécution des tâches qui lui sont confiées et d'envoyer périodiquement des informations, concernant l'état des tâches en cours d'exécution et la charge de la machine, au daemon *sge_ qmaster*.

3.3.4 Legion

3.3.4.1 Caractéristiques :

Legion [52] [21] [10] est un environnement de méta-computing orienté objet qui se positionne au dessus des systèmes d'exploitation existants. C'est un système flexible qui s'adapte à la politique de placement de chaque machine. Legion répond à dix objectifs : autonomie des sites (chaque machine contrôle ses ressources), extensibilité du coeur du système (tous les composants de Legion sont extensibles et remplaçables), extensibilité de l'architecture, facilité d'utilisation (la complexité est masquée à l'utilisateur), utilisation du parallélisme pour la haute performance, stockage des données dans un annuaire unique, prise en compte de la sécurité, gestion de l'hétérogénéité des ressources, support de plusieurs langages pour les applications, tolérance aux fautes.

Le système Legion est constitué de plusieurs objets hiérarchiques : un objet représentant les capacités de la machine et utilise la réservation (*Host object*), un objet représentant le stockage d'informations (*Vault object*) sauvegardant l'état persistant d'un objet Legion.

3.3.4.2 Architecture :

Le modèle de gestion de ressources contient les ressources (*Hosts et Vaults*), la base de données d'informations statiques (*Collection*), le scheduler (*Enactor*) et un objet réalisant l'exécution (*execution Monitor*). Dans l'implémentation actuelle, il n'y a pas d'objet séparé réalisant le monitoring : l'*Enactor* et le *Scheduler* réalisent cette tâche. Trois groupes de fonctions sont possibles pour les objets de type *Host* : la gestion de réservation (Unix n'a pas de notion de réservation donc un objet standard Unix de type *host* maintient une table de réservation dans le *Host objet*), gestion des objets et diffusion de l'information.

3.3.4.3 Placement :

Le placement est une négociation entre des agents agissant au nom des applications et des ressources. Le placement proposé est un placement aléatoire où les dépendances entre objets ne sont pas considérées.

3.3.4.4 Interaction avec un client :

Quand un client demande un placement pour une application, différentes étapes sont suivies. Le *Scheduler* interroge le *Collection* pour obtenir de l'information sur les ressources, envoie l'ordonnancement à l'*Enactor* qui négocie l'instanciation avec les objets ressources. L'*Enactor* fait les réservations, renvoie les résultats au *Scheduler* et attend son approbation. Ensuite, il gère le placement sur les machines et supervise le statut des travaux en cours. L'*Enactor* peut réaliser une co-allocation. L' *Execution Monitor* peut demander de recalculer un ordonnancement. Le modèle de Legion et les étapes suivant une requête d'un client sont décrits sur la figure 3.8.

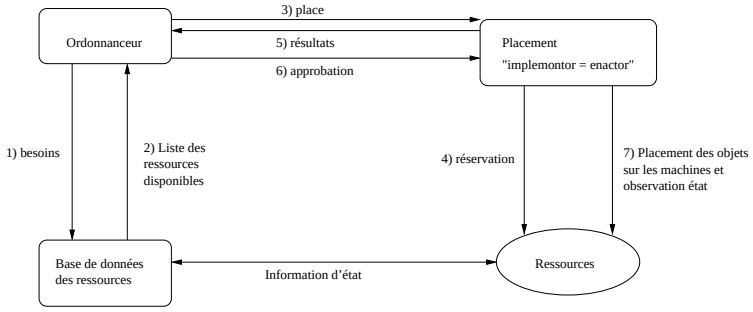


Fig. 3.8 – Modèle de Legion

3.4 Environnements client/serveur pour les Grilles

Dans cette section, les intergiciels dont le but est de fournir l'accès à des serveurs, situés dans des domaines administratifs différents, en utilisant le paradigme classique d'appel de méthodes distantes (RPC) sont décrits. Ces outils semblent être regroupés depuis peu dans la littérature sous le terme de *Network Enabled Servers*, et utilisent un standard de communication appelé *GridRPC* [19].

3.4.1 Ninf :

Ninf [54][51] permet à ses utilisateurs d'accéder à des ressources matérielles, logicielles et à des données scientifiques distribuées à partir d'une interface conviviale. L'accès aux serveurs de calculs distants se fait de manière transparente en utilisant le paradigme des appels de méthodes distantes (RPC). Les données de l'application (matrices, fichiers...) sont transmises au serveur distant de manière efficace, le serveur exécute le traitement et renvoie le résultat du calcul. Les avantages de Ninf sont les suivants :

- permet de déporter les parties les plus gourmandes en calcul d'une application vers plusieurs serveurs de calculs haute performance distants. Ceci est réalisé sans utilisation de matériel ou de logiciel particulier.
- l'API de Ninf est conçue de manière à être conviviale et familière d'utilisation pour des habitués de la programmation en C, C++ et FORTRAN. Le client dialogue avec les bibliothèques distantes de Ninf sans avoir besoin d'aucune connaissance en programmation réseau.
- Les appels de méthodes distantes de Ninf peuvent être asynchrones et automatiques : pour les applications parallèles, un composant *Ninf metaserver* conserve des informations sur l'état des serveurs Ninf sur le réseau et répartie automatiquement les appels de méthodes distantes sur les serveurs adéquats afin de répartir la charge.
- Ninf possède un composant chargé d'interroger régulièrement des bases de données contenant les valeurs précises de constantes scientifiques importantes (par exemple en physique et en chimie) afin d'éviter à l'utilisateur de devoir fournir des valeurs approchées de ces constantes.

Une interface WEB permet de lister, trier ou d'utiliser (il y a donc deux manières d'utiliser les bibliothèques : par appel de fonction dans les programmes ou par le WEB) les applications enregistrées sur les serveurs. La machine serveur et le client peuvent être connectés à travers un réseau local ou via Internet, les machines peuvent être hétérogènes : les données sont transmises lors de communications dans un format de données commun.

3.4.1.1 Création d'un composant Ninf

Le portage d'un code de calcul existant sous Ninf est réalisé par le *library provider* en 5 étapes :

1. l'API du code de calcul est décrite en langage *Ninf IDL* (Interface Description Language)
2. le fichier *Ninf IDL* obtenu est compilé afin de générer les fichiers d'en-tête et le code *stub* pour la nouvelle application
3. le code de calcul est compilé sur l'architecture du serveur distant
4. le *linkage* du code obtenu avec les bibliothèques RPC de Ninf est réalisé
5. le code obtenu est enregistré sur le serveur Ninf s'exécutant sur la machine.

Certaines applications populaires comme LAPACK (paquetages de bibliothèques d'algèbre linéaire) sont déjà disponibles sous cette forme. Les communications entre client et serveur se font en utilisant des connexions TCP/IP afin de garantir une communication sûre entre les processus. Dans un environnement hétérogène, Ninf utilise le format de données XDR comme protocole par défaut.

3.4.1.2 Connexion à un serveur Ninf

Le client peut spécifier le serveur auquel il souhaite se connecter de différentes manières :
- dans le fichier .ninfrc
- dans une variable d'environnement
- avec une URL
- allocation automatique par le *Ninf Metaserver*.

La procédure est la suivante : le client interroge le serveur afin d'obtenir la description de l'application qu'il souhaite utiliser, en fonction de la description renvoyée par le serveur, le client envoie les arguments et récupère par la suite ses résultats. Le Ninf *Metaserver* est une sorte d'annuaire des serveurs Ninf (adresse, numéro de port, liste de fonctions enregistrées sur le serveur, la distance avec le serveur en respectant la bande passante, la possibilité de calcul de la machine, le statut du serveur et la charge) et permet au client de choisir un serveur approprié. Le client peut interroger le *metaserver* et déléguer le problème. Utiliser le *metaserver* permet d'assurer l'équilibrage de charge (le *metaserver* trouve le meilleur serveur en fonction des ressources et de la charge). L'infrastructure de Ninf comporte plusieurs *metaservers* et serveurs. Les *metaservers* échangent des informations périodiquement.

Le travail futur autour de Ninf : authentification, comptabilité (facturation), exportation de code client (pour l'instant rien n'est offert), tolérance aux fautes (l'objectif est de récupérer les transactions lors de panne réseau) et sécurité (les objectifs sont de sécuriser les nœuds serveurs et de crypter les données).

3.4.2 NetSolve :

Netsolve [11][39] est un système client-agent-serveur permettant d'accéder à des ressources logicielles et matérielles distantes à partir de divers environnements scientifiques tels que Matlab, Mathematica et Octave ou à partir de langages de programmations courants tels que le C ou le FORTRAN. Netsolve est composé de trois entités :
- Le client qui a besoin d'exécuter des appels de méthodes distantes. Le client peut être invoqué à partir d'une application existante (Matlab...) ou d'un programme spécifique.
- Le serveur qui exécute des tâches au nom des clients. Les serveurs peuvent être hébergés par une simple machine mono-processeur ou par une machine parallèle. Les serveurs Netsolve s'exécutent sur leur machine en concurrence avec les autres applications de la machine.
- L'agent est le point névralgique de Netsolve. Il gère la liste de tous les serveurs disponibles, sélectionne les ressources pouvant répondre aux requêtes des clients et assure le partage de charge entre les serveurs.

En pratique, du point de vue du client, la connexion à un serveur distant est réalisée de manière totalement transparente. Cependant, une telle opération met en jeu les étapes suivantes :

1. le client interroge l'agent afin d'obtenir le référence d'un serveur pouvant lui fournir la fonction désirée.
2. l'agent renvoie une liste de serveurs triée par ordre de convenance.
3. le client essaie de contacter un serveur de la liste, en commençant par le premier et en décalant en cas d'échec. Le client envoie ensuite les données de son problème au serveur ayant répondu.
4. le serveur exécute le traitement au nom du client et renvoie le résultat.

En plus de fournir l'intergiciel nécessaire à l'exécution de la requête distante, Netsolve fournit des mécanismes afin de s'interfacer avec d'autres services de gestion de grilles. Ceci est réalisé de deux manières, soit en utilisant un client capable de dialoguer avec le service de gestion de grille externe (utilisation du standard naissant GridRPC [19]), soit en utilisant un serveur servant de passerelles entre le client Netsolve et le service externe. L'utilisation de passerelles est particulièrement utile afin de permettre l'utilisation de mécanisme tels que MPI ou Condor tout en conservant l'utilisation ordinaire de Netsolve.

3.4.2.1 Gestion de ressources

Les *agents* de Netsolves utilisent la connaissance qu'ils ont sur l'application, sur la taille des paramètres fournis par le client et sur l'état courant des ressources afin de proposer une liste de serveurs adéquats permettant de répondre à la requête. Lors de la création d'un nœud Netsolve, le serveur créé fournit à l'agent la liste des services qu'il propose ainsi que la complexité combinatoire de ces services. Cette complexité est fournie sous la forme de deux entiers a et b et évaluée sous la forme aN^b, où N représente une expression de la taille du problème. L'agent possède une connaissance de la capacité de calcul (MFlops) de chaque serveur (fournie lors de la création du serveur) ainsi que de la charge courante de chaque serveur (envoyée régulièrement par le serveur). La bande passante et la latence entre le serveur et l'agent sont également mesurée et servent d'approximation afin d'exprimer la capacité de communication entre un client et le serveur. Lorsque l'agent reçoit une requête pour un traitement d'une taille donnée, il utilise sa connaissance de la complexité du problème à résoudre, de la charge du serveur et de la capacité de communication du serveur afin de calculer la date de terminaison de la tâche sur chaque serveur et de trier les serveurs par date de terminaison croissante. La requête du client est ensuite traitée par le serveur offrant la date de terminaison la plus proche. En cas de défaillance du serveur en question (panne matérielle, logicielle ou panne réseau), la requête est automatiquement soumise au deuxième serveur de la liste et ainsi de suite, soit jusqu'à la complétion du traitement, soit jusqu'à l'épuisement de tous les serveurs de la liste.

Pour les applications parallèles, une étude du flux de données entre les tâches est réalisée afin de limiter les échanges de données entre le client et les serveurs. Une étude préliminaire des données d'entrées et de sorties de l'ensemble des tâches composant l'application parallèle est réalisée afin de créer un graphe acyclique de dépendances entre tâches. Ce graphe, ainsi que toutes les tâches concernées sont ensuite envoyées vers le même serveur Netsolve, afin de conserver les données sur ce serveur jusqu'à la terminaison de l'application parallèle.

3.4.3 DIET

DIET (Distributed Interactive Engineering Toolbox) est un ensemble hiérarchique de composants utilisés pour le développement d'applications basées sur des serveurs de calcul sur la grille. Le but du projet DIET est de fournir une boîte à outils permettant le portage d'un grande diversité d'applications sur la grille. Ce projet est implémenté en C++ et utilise Corba comme intergiciel de communication.

3.4.3.1 Architecture

L'architecture de DIET est basée sur une approche hiérarchique afin de faciliter le passage à l'échelle. DIET repose sur plusieurs composants. Un client est une application qui utilise DIET pour résoudre un problème en utilisant une approche RPC. Les utilisateurs peuvent se

connecter à DIET de différentes manières : via un portail WEB, en utilisant des environnements tels Scilab ou via des programmes écrits en C ou en C++. Des SeD (ou daemon serveur) servent d'interface entre le client et les serveurs de calcul et peuvent fournir autant de services de calcul que nécessaire. Un SeD peut servir d'interface d'exécution vers une machine unique ou vers une machine parallèle au quel cas il transmettra les requêtes de soumissions à un ordonnanceur effectuant du traitement par lots. Les agents fournissent des services de niveau supérieur tels que l'ordonnancement et la gestion des données. Ces composants sont organisés de manière hiérarchique de la manière suivante : un unique Master Agent (MA), plusieurs Agents (A) et des Local Agents (LA) (voir figure 3.9).

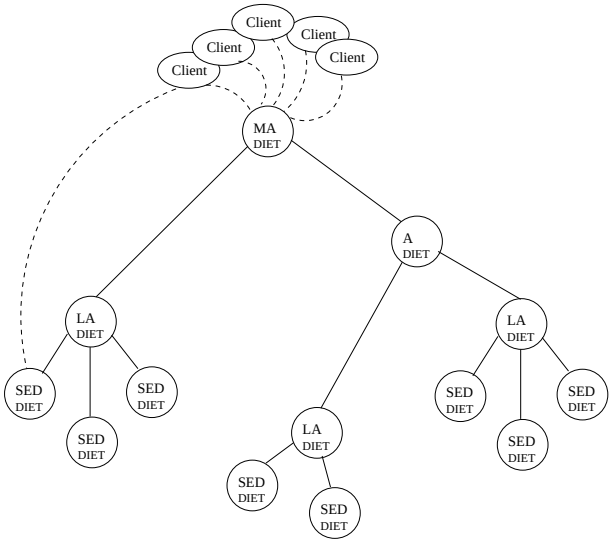


Fig. 3.9 – Architecture de DIET

Un Master Agent est le point d'entrée de l'environnement. Les clients se connectent au MA via le service d'annuaire de Corba en fournissant le nom du MA auquel ils souhaitent accéder. Ils soumettent ensuite leur requête au MA, qui la propage dans l'arbre des agents jusqu'à ce quelle aboutisse aux SeDs. Chaque SeD évalue alors sa propre capacité à fournir le service demandé, cette capacité peut dépendre de plusieurs critères tels que la charge du serveur, la localité des données ou la prédiction de la durée du service demandé... Chaque SeD renvoie ensuite sa réponse via la hiérarchie d'agents. Chaque agent agrège les réponses de ses fils et propage l'information jusqu'à ce qu'elle arrive au MA. Ce dernier envoie alors au client une liste de serveurs capables de répondre à la requête, cette liste est ordonnée selon une fonction objectif (durée de calcul, durée de communication, charge machine...). Le client choisit ensuite le serveur de la liste vers lequel il souhaite envoyer sa requête (généralement le premier).

3.4.3.2 Gestion de ressources et ordonnancement

Afin de réaliser un bon ordonnancement des tâches il est nécessaire de mettre en adéquation les besoins des tâches avec les ressources système disponibles. Les besoins des applications s'expriment en termes de temps d'exécution, d'espace mémoire nécessaire et de taille des données échangées sur le réseau. Les ressources système disponibles concernent le nombre de machines, leur puissance ainsi que les caractéristiques du réseau d'interconnexion. DIET utilise la bibliothèque FAST dont le but est de fournir une interface standard pour obtenir des informations

sur les besoins des applications et des informations sur l'état des ressources disponibles, et ce indépendamment de la manière dont ces information sont collectées. FAST utilise deux outils, un outils de collecte d'informations, et une base de données destinée à contenir des informations sur les besoins des applications, afin de renseigner ses clients. FAST permet de *plugger* facilement n'importe quel outils fournissant l'un des deux services précédemment cités. DIET utilise NWS comme outils d'observation et une base de donnée renseignée par une suite de benchmarks afin de connaître les besoins des applications : lors de l'installation de FAST, une suite de tests de performances sont réalisés sur les applications auxquelles DIET donne accès, les informations recueillies durant ces tests sont ensuite stockées dans une base de données afin d'être utilisées par la suite lors d'une décision de placement.

Le modèle d'appel de méthodes distantes suppose que le client transmette les données d'entrée nécessaires à l'exécution de son application lors de la soumission de sa requête. Ce modèle peut s'avérer pénalisant lorsque des données de taille importante doivent être utilisées par plusieurs requêtes successives d'un même client. En effet, les données seront transmises plusieurs fois entre le client et la grappe de machine provoquant des temps de communication inutiles. DIET fournit des mécanismes afin d'éviter ces transfert inutiles. Le client peut, lors de la soumission d'une requête, indiquer si les données qu'il fournit doivent être conservées ou non après l'exécution de sa requête.

3.5 Conclusion

Les principaux outils permettant de traiter le problème de la gestion de ressources sur grappes et grilles de machines ont été présentés. Certains de ces outils n'adressent qu'une partie du problème (Ganglia, NWS), d'autres ont pour but d'offrir un ensemble d'outils permettant de recouvrir tous les types d'utilisation de la grille (Globus), d'autres encore focalisent leurs efforts sur une utilisation précise des grappes (NINF, Netsolve) ou sur les services permettant une exploitation commerciale de la grille (Sun GridEngine).

Les différents projets présentés précédemment l'ont été dans leur dernière version connue en 2005. La plupart de ces outils ont connu des évolutions importantes ou ont vu le jour parallèlement à cette thèse. Ainsi, le projet Ganglia débutait lorsque nous avons commencé nos travaux et Sun GridEngine a subi d'importantes évolutions destinés à raffiner sa gestion des ressources et son modèle économique. Les notions de Grid Services et de GridRPC sont également apparues au cours des deux dernières années, fruit des efforts de standardisation menés par le Global Grid Forum.

Chapitre 4

La gestion de ressources dans Aroma.

Sommaire

4.1 Introduction

Le gestionnaire de ressources Aroma (scAlable ResOurce Manager and wAtcher), un des résultats de cette thèse va maintenant être présenté. Ce gestionnaire de ressources a été développé dans le cadre du projet RNTL CASP (*Clusters for Application Service Provider*[9] [18]

[59]) qui visait la création de services permettant de déporter l'exécution d'applications industrielles, gourmandes en calculs, sur des grappes de machines. L'ambition de ce projet était de permettre aux entreprises d'exécuter leurs applications sur une grappe de machines via une interface simple et conviviale, tout en fournissant l'infrastructure nécessaire à la facturation de ce service auprès des utilisateurs : gestion fine des ressources permettant de garantir différents niveaux de qualité de service, facturation. Un des enjeux de ce projet était de développer de nouveaux algorithmes de gestion de ressources traitant de la notion de qualité de service.

Aucun des systèmes de gestion de ressources présentés précédemment n'offrait de réponse satisfaisante aux problèmes posés par le projet CASP. La plupart ne permettaient pas d'observer l'état de l'ensemble des ressources nécessaires aux algorithmes de placement développés ou n'offraient qu'une vision trop imprécise de cet état. D'autres encore n'offraient pas l'infrastructure nécessaire à la mise en place de modèles économiques ou n'offraient pas de garantie suffisante en termes de pérennité du système. Ces constatations ont amené à développer un nouveau gestionnaire de ressource, dont l'architecture va maintenant être présentée.

4.2 Positionnement du problème

Le schéma d'utilisation du gestionnaire Aroma (figure 4.1) est le suivant. Des fournisseurs de services possèdent une ou plusieurs grappes de machines sur lesquelles sont installés différents noyaux de calculs d'applications demandant une importante capacité de traitement. L'utilisateur de ces applications peut, soit continuer à utiliser son application de manière traditionnelle, en exécutant les calculs sur sa machine locale soit, dans le cas de traitements coûteux en calcul, choisir de déporter le traitement sur les grappes de machines du fournisseur de services. Le client peut utiliser Aroma de deux manières différentes : soit par l'intermédiaire d'une interface graphique, soit directement à partir de son application en faisant appel à l'API d'Aroma. Dans les deux cas, la requête est transférée au fournisseur de services via le réseau (réseau local ou l'Internet). Cette transmission se fait en utilisant des mécanismes d'authentification et de chiffrement afin de préserver la confidentialité des informations émises. La requête de calcul, ainsi que les données nécessaires à la réalisation du calcul sont alors traitées par Aroma chez le fournisseur de services. Ce dernier exécute le noyau de calcul correspondant à l'application sélectionnée, sur la ou les machines choisies par un algorithme de placement. Une fois le traitement achevé, Aroma en informe le client, qui peut alors récupérer ses résultats.

Aroma se positionne dans la catégorie des *Network Enabled Servers*, il n'utilise pas pour l'instant le standard GridRPC, mais repose sur le paradigme d'appels de méthodes distantes. L'intergiciel de communication utilisé est Jini [1] qui sera présenté par la suite.

Aroma s'est focalisé dès sa création sur les problématiques suivantes :

- *Gestion fine des ressources :* l'ordonnancement des tâches est soumis à certaines contraintes, notamment celle de garantir un certain niveau de qualité de service vis-à-vis de l'utilisateur. Afin d'atteindre cet objectif, l'ordonnanceur doit pouvoir s'appuyer sur une connaissance détaillée et actualisée de l'état des ressources de la grappe de machines.

[1] Jini and all Jini-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

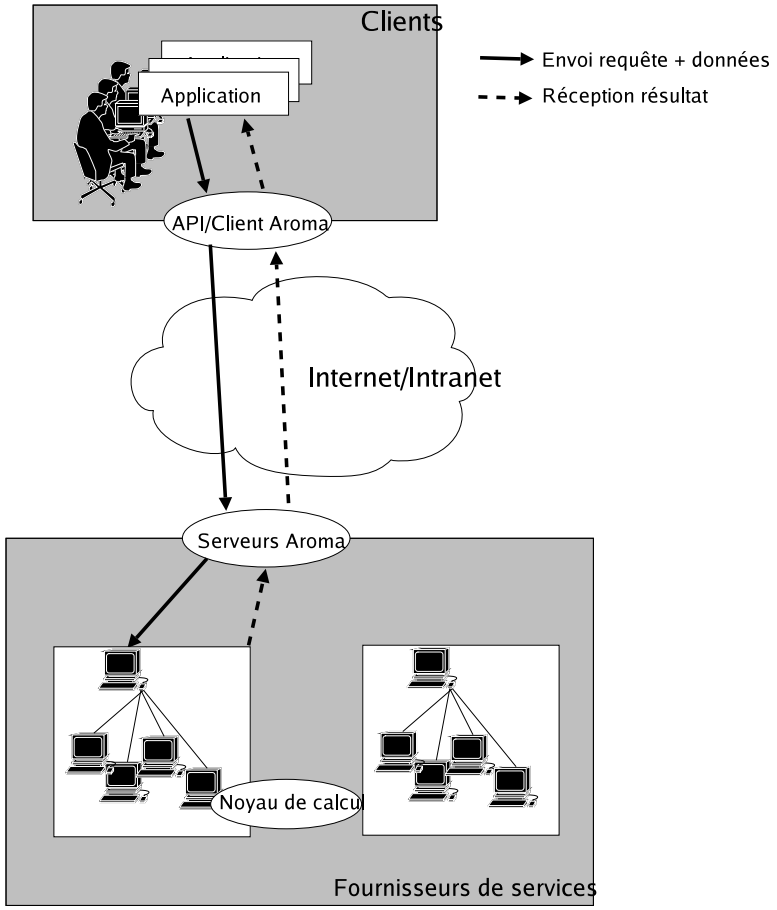


Fig. 4.1 – Schéma d'utilisation d'Aroma

– *Prise en compte de l'aspect dynamique des ressources :* une grille ou un ensemble de grappes de machines impliquent un grand nombre de nœuds. Dans de tels environnement, la défaillance ou l'ajout d'un nœud sont des événements courants qui ne doivent pas remettre en cause le fonctionnement de la structure. Pour la même raison, l'ajout de nouvelles fonctionnalités au gestionnaire doit pouvoir se faire, dans la mesure du possible, sans qu'il soit nécessaire d'arrêter le service et de re-déployer le gestionnaire sur l'ensemble des machines.
– *Facturation du service :* une trace de l'utilisation des ressources doit être conservée afin de développer des modèles économiques autour des grilles, dans le but de facturer l'utilisation du service au client.

Nous allons, dans un premier temps, présenter l'architecture générale d'Aroma. Nous détaillerons par la suite le modèle de communication employé par le gestionnaire. Enfin, les fonctionnalités des différents composant du gestionnaire seront abordées.

4.3 Architecture générale du gestionnaire de ressources

4.3.1 Les différents niveaux hiérarchiques

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

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

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

4.3.2 Architecture logicielle

Un serveur Aroma est présent à chaque niveau hiérarchique : machine, grappe, domaine et grille. Chaque serveur Aroma est un service Jini (voir section 4.4), spécialisé en fonction du niveau hiérarchique qu'il représente. Plusieurs composants sont regroupés au sein d'un serveur Aroma (figure 4.2).

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

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

Les spécificités de traitement liées à chaque niveau hiérarchique vont maintenant être présentées.

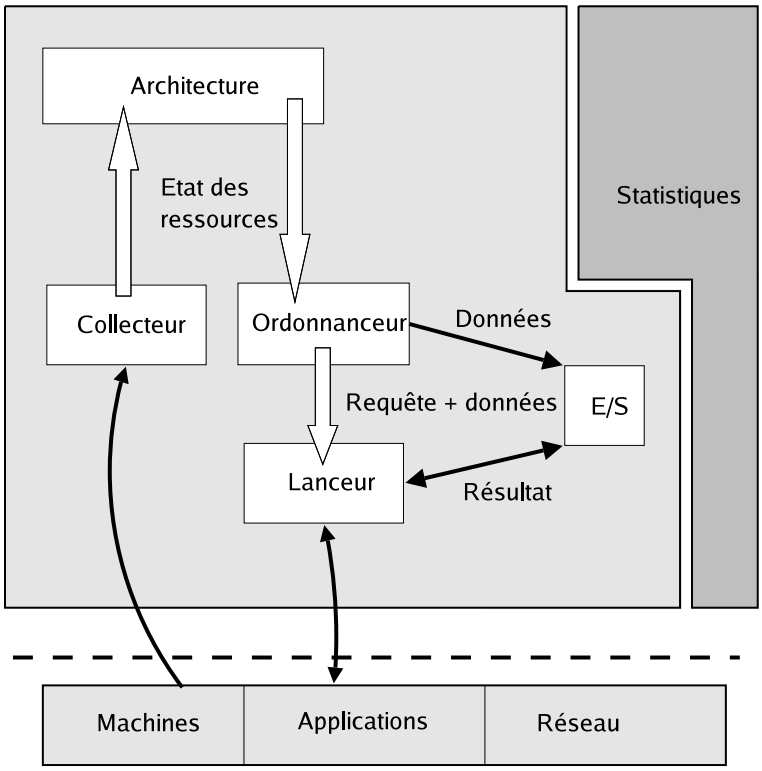


Fig. 4.2 – Serveur générique

Niveau machine : Un serveur de niveau machine appelé *Host Resource Unit* (HRU) est présent sur chaque machine de la grille. Il collecte toutes les informations disponibles au niveau d'une machine : charge de la machine (processeur, mémoire, etc), charge du réseau vue par la machine, les applications et processus s'exécutant sur la machine et les utilisateurs de la machine. Il n'y a pas d'ordonnanceur à ce niveau, le «lanceur» exécute les ordres provenant des ordonnanceurs de niveau supérieur.

Niveau grappe de machines : Une machine parmi celles constituant la grappe héberge le service de niveau grappe appelé *Cluster Resource Unit* (CRU). Le CRU possède une connaissance agrégée de l'état de la grappe (figure 4.3). Des décisions de placement peuvent être prises à ce niveau.

Niveau domaine : Comme pour le niveau grappe, une machine appartenant au domaine héberge le service de niveau domaine appelé *Domain Resource Unit* (DRU). Le domaine est à priori géré par un administrateur unique. On retrouve les mêmes fonctionnalités que sur la grappe, simplement les informations recueillies seront encore plus synthétiques (figure 4.3).

Niveau grille : Les entités de niveau grille appelées *Grid Resource Unit* (GRU) représentent des points d'accès à un ensemble de domaines et sont localisées sur ces différents domaines. Le GRU possède une vue très agrégée de l'état de la grille (figure 4.3). L'ordonnanceur d'un GRU peut, soit prendre une décision de placement, soit transférer la requête vers un de ses domaines qui se chargera de placer l'application sur ses machines.

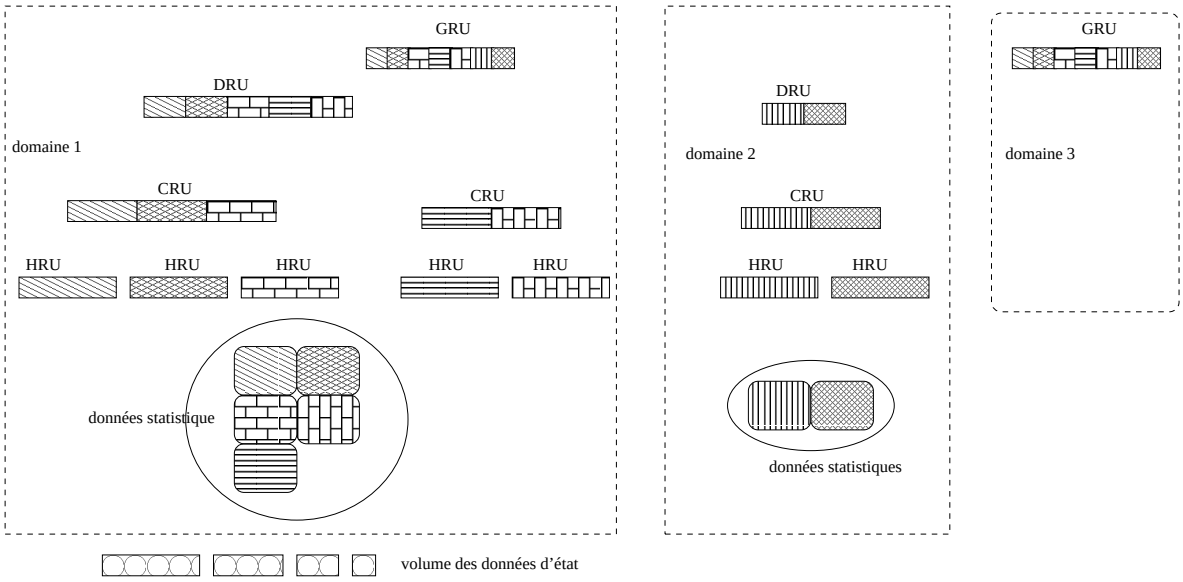


Fig. 4.3 – Gestion de l'information

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

4.3.3 Exemple d'architecture

La figure 4.4 présente un exemple d'une architecture complète. On retrouve un HRU sur chaque machine destinée à servir de support d'exécution aux applications. Par contre, certaines machines n'hébergent que des CRU, DRU ou GRU. Ceci dépend de la configuration du site faite par l'administrateur. Certaines machines cumulent plusieurs rôles.

4.4 Système de communication

Aroma repose sur l'utilisation de la technologie Jini [38], développée par SUN [67]. Jini définit plusieurs protocoles de communication permettant à un ensemble de services de se fédérer et d'interagir au sein d'un réseau. Ces services, qui peuvent être indifféremment des dispositifs matériels ou logiciels, s'auto-détectent et coopèrent de manière dynamique et robuste. Nous présentons tout d'abord les concepts de base de Jini, avant d'en détailler l'utilisation dans le cadre d'Aroma.

4.4.1 Principes de base de Jini

Jini repose principalement sur un protocole de *discovery*, un protocole de *join*, un protocole de *lookup* et un composant particulier appelé *Lookup service* [44].

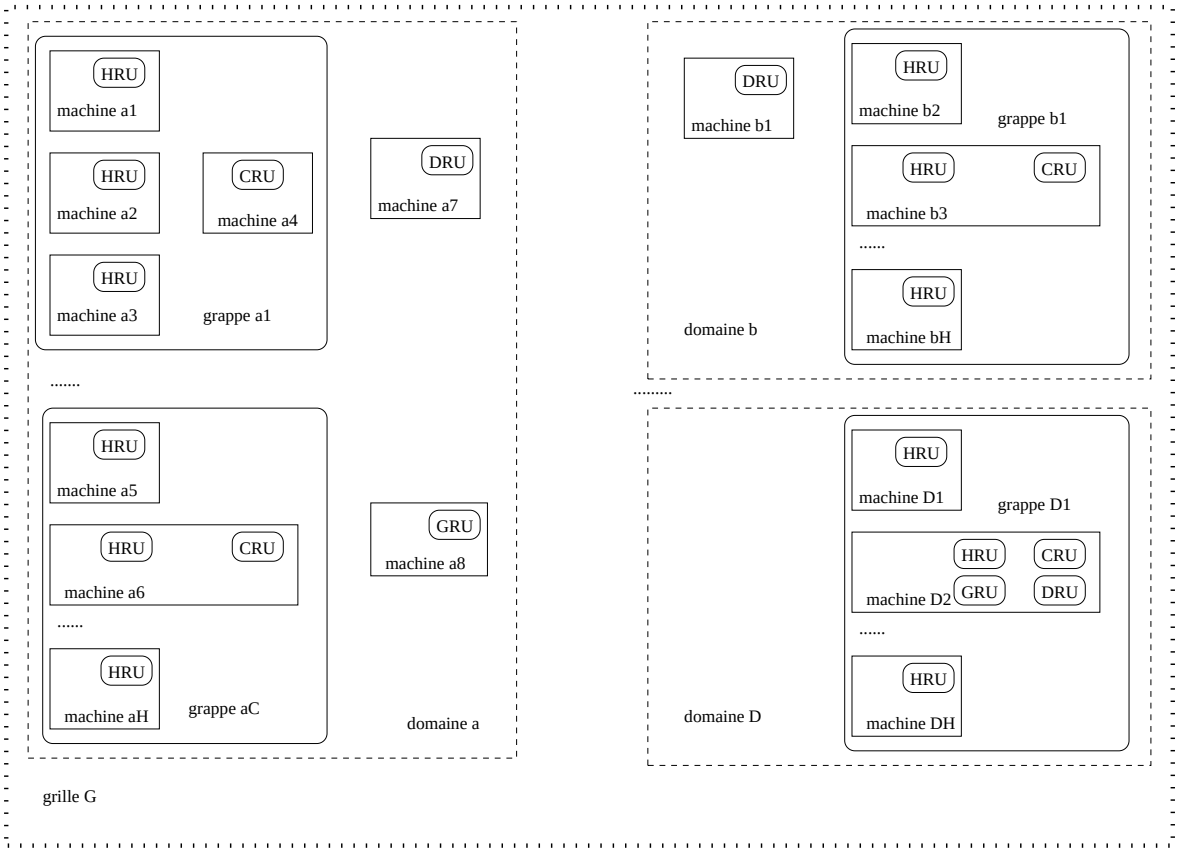


Fig. 4.4 – Exemple d'architecture

4.4.1.1 *Lookup service.*

Le *lookup service* est un annuaire de tous les services Jini accessibles sur une partie du réseau. Il maintient une description ainsi qu'une référence vers chaque service. Jini n'impose aucune restriction sur le moyen de communication utilisé entre un client et un service. Le client peut, soit exécuter le service directement sur sa machine (au quel cas la référence du service est la routine de code à exécuter), soit dialoguer avec le service distant en utilisant des appels de méthodes distantes (au quel cas la référence du service est le *stub*). La seule restriction concerne le dialogue entre le client et le *Lookup service* qui repose sur l'utilisation de RMI[2].

4.4.1.2 *Discovery.*

Cette action est réalisée lorsqu'un client ou un service cherche à entrer en contact avec un ou plusieurs *lookup services*. La phase de *discovery* utilise la notion de groupes. Chaque *lookup service* gère un ou plusieurs groupes et ne connaît que les services appartenant à son groupe. La phase de *discovery* peut utiliser trois protocoles : *multicast request protocol*, *multicast announcement protocol* et *unicast discovery protocol*.

– *multicast request protocol* : un service souhaitant contacter des *lookup services* envoie

[2] `http://java.sun.com/products/jdk/rmi/`

des paquets multicast sur le port 4160 du groupe de multicast 224.0.1.85.
Le service envoie une requête contenant son adresse IP, le numéro de port sur lequel il écoute, l'ensemble des groupes qu'il souhaite contacter et les identifiants des *lookups services* dont il a déjà connaissance.
Chaque *lookup service* intercepte les paquets multicast, regarde s'il gère les groupes demandés et s'il n'est pas déjà connu. S'il est concerné par la requête, il transmet sa référence (dialogue TCP) au service client.

- *multicast announcement protocol* : ce protocole est utilisé par un *lookup service* nouvellement créé afin de prévenir les services existants de sa naissance.
  Le *lookup service* envoie périodiquement des paquets sur le port 4160 du groupe de multicast 224.0.2.84. Ces paquets contiennent, entre autres, les informations nécessaires pour le contacter en utilisant une requête TCP.
- *unicast discovery protocol* : ce protocole permet de contacter directement un *lookup service* dont on connaît l'adresse. Les communications multicast possèdent l'inconvénient majeur d'être souvent bloquées par les routeurs. Il est donc utile de posséder un protocole permettant de contacter n'importe qu'elle machine de l'Internet : c'est le rôle de l'*unicast discovery protocol*. Ce dernier permet de réaliser une communication TCP avec n'importe quelle machine dont on possède l'adresse IP.

4.4.1.3 *Join.*

Le protocole de *join* permet à un service de s'enregistrer auprès d'un *lookup service*. Une fois le *lookup service* découvert grâce au protocole de *discovery*, le service envoie sa description et sa référence au *lookup service* en utilisant le protocole de *join*.

4.4.1.4 *Lookup.*

Cette action est réalisée par un client souhaitant utiliser un service, afin de récupérer la référence du service et de dialoguer avec ce dernier. Le client contacte un ou plusieurs *lookup services* en utilisant un des protocoles de *discovery* décrit précédemment. Une fois un *lookup service* contacté, le client fournit une description du service avec lequel il souhaite interagir. Si le *lookup service* connaît un service correspondant à la description, il renvoie la référence du service en question.

4.4.1.5 Les *Leases.*

Un *lease* (bail) représente le droit d'utilisation d'un service pour une durée donnée. Lorsqu'un service s'enregistre auprès d'un *lookup service*, il utilise un *lease*. Si le service n'a pas renouvelé son *lease* avant l'expiration de ce dernier, le *lookup service* supprimera la référence du service de son annuaire. Le mécanisme de *leasing* peut également être utilisé entre un client et un service, le client ne peut alors utiliser le service que s'il possède un *lease* en cours de validité. Ceci est utilisé, notamment lorsque le service stocke des données pour le client, afin d'éviter la conservation de données inutiles.

4.4.1.6 Tests de performance

Nous avons réalisé le test de performance suivant afin d'évaluer la consommation (processeur et mémoire) d'un service Jini.

Description du test

Nous utilisons un service Jini qui crée un thread calculant la charge du processeur toutes les 5 secondes (boucle infinie). L'observation est réalisée par une routine écrite en langage C. Le client interroge le service toutes les 6 secondes afin de simuler le taux de rafraîchissement d'une interface graphique. Le service utilise le *lookup discovery service* pour rechercher les lookup services et le *lease renewal service* pour gérer ses *leases*.

Conditions du test

- la version de Jini est la version 1.1
- les services Jini sont exécutés sur des machines distinctes
- les machines utilisées sont des SunBlade100 : 500 Mhz, 640Mo de RAM.
- durée du test : 2 heures
- la consommation cpu est évaluée avec la commande *time* d'unix
- la consommation mémoire est évaluée grâce à la commande *top*

Résultats obtenus

Machine	Nom du programme	%cpu	Mémoire	Résidant
1	serveur HTTP	0.0%	42Mo	19Mo
2	rmid	0.4%	34Mo	12Mo
2	lookup Service	0.0%	42Mo	19Mo
3	rmid	0.5%	34Mo	12Mo
3	L.Discovery Service	0.0%	43Mo	21Mo
3	Lease Renewal Serv.	0.0%	42Mo	20Mo
4	serveur HTTP	0.0%	42Mo	19Mo
4	service	0.1%	41Mo	19Mo
5	client	0.1%	40Mo	18Mo

Jini est très peu gourmand en temps processeur, par contre il utilise beaucoup de mémoire. Ceci est du à la machine virtuelle Java, en effet un simple programme Java affichant "Hello" consomme 34Mo de mémoire dont 12Mo en résident.

4.4.2 Le service Jini Aroma

L'ensemble des fonctionnalités du gestionnaire de ressources sont regroupées au sein d'un même service Jini qui s'enregistre auprès d'un ou plusieurs *lookup services* (figure 4.5)[61]. Un *lookup service* est positionné sur chaque grappe, chaque domaine et sur chaque machine hébergeant un service GRU. Chaque *lookup service* gère un groupe correspondant au nom de sa grappe (respectivement domaine, grille). Dans le cas d'une machine jouant plusieurs rôles, un seul *lookup service* gérant les groupes associés à chaque rôle est utilisé.

Lors de son enregistrement auprès d'un *lookup service*, chaque service Aroma fournit un nom permettant de l'identifier ainsi qu'une information sur son rôle (HRU, CRU, DRU ou GRU).

Un client souhaitant se connecter au gestionnaire de ressources peut le faire de deux manières :

- soit le client connaît l'adresse IP de la machine hébergeant le service Aroma. Dans ce cas, il utilise une requête unicast en indiquant l'adresse de la machine, le nom et le rôle

du service auquel il souhaite se connecter. Ce mode de connexion est notamment le seul utilisable lorsque l'on souhaite se connecter à un service appartenant à un autre domaine d'administration réseau (au niveau grille notamment).

– soit le client émet une requête multicast en indiquant le nom de la grappe ou du domaine, le nom et le rôle du service recherché.

Les services Aroma sont implémentés en utilisant RMI. De ce fait, les *lookups services* fournissent au client un *stub* permettant de dialoguer directement avec le serveur en invoquant des méthodes distantes.

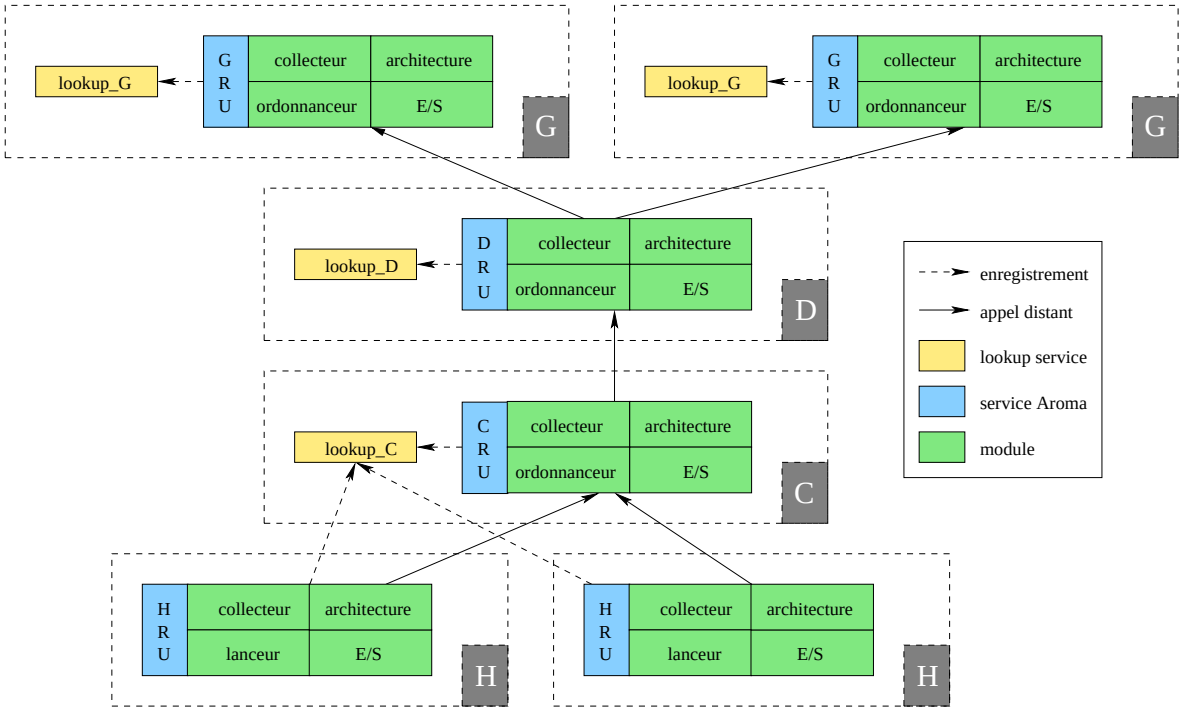


Fig. 4.5 – Jini et Aroma

4.4.3 Communications entre *Resource Unit*

Nous allons maintenant détailler les échanges intervenants entre les différents niveaux hiérarchiques de la grille lors de sa création, puis de son exploitation.

4.4.3.1 Niveau grille

Les différents services GRU doivent avoir la même vue globale de la grille, et chaque *lookup service* du groupe grille doit connaître tous les services GRU de la grille.

Lors de sa création, un service GRU reçoit la liste des adresses IP de tous les *lookup services* de niveau grille déjà existants. Par la suite, le service GRU nouvellement créé s'enregistre auprès de tous ces *lookup services*. Il récupère par la même occasion les références de tous les services GRU enregistrés auprès de ces *lookup services*. Il peut alors contacter ces différents services GRU afin de les informer de son existence, et de partager sa connaissance de l'architecture de la grille.

4.4.3.2 Niveau domaine

Un service DRU nouvellement créé doit connaître le nom du groupe de *lookup services* auprès duquel il doit s'enregistrer ainsi que l'adresse IP d'au moins une machine hébergeant un *lookup service* de niveau grille.

Le service s'enregistre auprès des *lookup services* de son groupe afin de pouvoir être contacté par des clients ou par d'autres services. Par la suite, il utilise la liste d'adresses IP qui lui a été fournie afin de s'enregistrer auprès de chaque service GRU dont il a connaissance.

Cet enregistrement possède une double utilité : il permet au GRU de bâtir sa vue hiérarchique de la grille, et il initie le processus d'échange de données d'état entre le DRU et le GRU. Lorsque le DRU s'enregistre auprès d'un GRU, ce dernier insère le DRU dans la liste de ses fils. Il contacte ensuite ce fils afin de lui demander de lui envoyer des informations concernant l'état courant de ses ressources. Ces données sont envoyées périodiquement sous forme d'événements. Un *lease* d'une durée égale à trois fois la période d'envoi des événements est associée à ces envois. L'expiration de ce *lease* permet de détecter un problème de communication entre le GRU et le DRU. Ce problème peut provenir d'une panne réseau, d'une défaillance matérielle sur la machine hébergeant le DRU ou d'une défaillance logicielle. Dans tous les cas, le service GRU supprime le nœud DRU associé de la liste de ses fils, afin de ne pas orienter un client vers une machine défaillante.

4.4.3.3 Niveau cluster

Lors de sa création, un service CRU reçoit le nom du groupe de *lookup services* auprès duquel il doit s'enregistrer ainsi que le nom du groupe de *lookup services* associé à son service DRU père.

Il interroge alors les *lookup services* du domaine afin de récupérer la référence de son service DRU père, et de s'enregistrer auprès de ce dernier. Comme pour le DRU, cet enregistrement possède la double utilité de mettre à jour la vision de la grappe qu'a le domaine, et d'initier l'échange d'informations d'état. Le remontée d'information est réalisée périodiquement et est associée à un *lease* comme pour le service DRU. La période de remontée d'information est cependant plus courte afin d'offrir une vision plus actualisée de l'état des ressources, et ainsi d'améliorer la qualité des décisions prises par l'ordonnanceur.

4.4.3.4 Niveau machine

Aucun *lookup service* n'est associé à un service HRU, ce dernier s'enregistre auprès des *lookup services* de sa grappe, et doit pour cela connaître le nom du groupe de ces *lookup services*. Le service HRU récupère la référence de son service CRU auprès du *lookup service* et sauvegarde cette référence dans son traitement architecture. Il contacte ensuite son CRU père, qui met à jour la liste de ses fils et formule une requête d'envoi d'informations d'état. Le service HRU envoie régulièrement des événements contenant l'état de la machine l'hébergeant.

4.5 Les différentes modules d'Aroma

Nous allons maintenant détailler le rôle et les fonctionnalités principales des différents modules d'Aroma : collecteur, lanceur, ordonnanceur.

4.5.1 Le module «collecteur» (*watcher*)

Ce module collecte l'information nécessaire à la connaissance de l'état des ressources. La difficulté est de collecter cette information sans surcharger exagérément les machines. Plusieurs types d'information sont nécessaires :

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

Le collecteur réalise deux traitements différents selon qu'il est utilisé au niveau machine ou à un niveau hiérarchique supérieur.

4.5.1.1 Niveau machine.

Les services HRU collectent directement les données d'état sur la machine les hébergeant. La collecte devant être peu gourmande en ressources, elle est réalisée par des routines écrites en langage C. Les données collectées sont ensuite communiquées au service HRU implémenté en Java en utilisant la JNI[3] *Java Native Interface*.

Les données statiques collectées sont les suivantes :

- Nom de la machine.
- Adresse IP de la machine.
- Le type et la version du système d'exploitation gérant la machine.
- Le type d'architecture de la machine (sparc, intel...).
- La fréquence et le nombre de processeurs de la machine.

Les données dynamiques collectées sont les suivantes :

- La charge processeur :
  - pourcentage d'utilisation des processus en mode *nice*
  - pourcentage d'utilisation des processus utilisateurs
  - pourcentage d'utilisation des processus systèmes

[3] `http://java.sun.com/docs/books/tutorial/native1.1/`

  - pourcentage inutilisé

  Une moyenne des dernières secondes d'utilisation de ses valeurs est calculée à partir des informations présentes dans le noyau.
- La charge mémoire :
  - La taille totale de la RAM
  - Le pourcentage de RAM utilisé
  - La taille totale de la mémoire SWAP
  - Le pourcentage de SWAP utilisé

  Ces informations sont également collectée dans le noyau du système d'exploitation.
- État des processus présents sur la machine :
  - Nombre de processus dans l'état *sleep*
  - Nombre de processus dans l'état *run*
  - Nombre de processus dans l'état *idle*
  - Nombre de processus dans l'état *zombi*
  - Nombre de processus dans l'état *stop*

  Ces informations sont également collectée dans le noyau du système d'exploitation.
- Ressources utilisées par les applications gérées par Aroma :
  - Temps processeur consommé par l'application,
  - Évolution de l'image mémoire de l'application.

La liste des ressources observées peut être enrichie dynamiquement, sans qu'il soit nécessaire d'arrêter le gestionnaire de ressources (voir section 4.6).

4.5.1.2 Niveaux supérieurs.

Le «collecteur» des services CRU, DRU et GRU se contente de calculer la moyenne des données dynamiques de chacun de ses fils. Par exemple, imaginons un service CRU recevant toutes les 30 secondes des informations d'état de la part de ses HRU fils. L'information qu'il reçoit représente la moyenne des valeurs collectées sur la machine sur les dernières 30 secondes. Le «collecteur» du CRU va calculer la valeur de la charge de sa grappe de machine en moyennant les valeurs de chaque HRU.

4.5.2 Représentation de l'état de la grille : le module architecture.

Les données d'état sont stockées sous la forme d'un arbre (figure 4.6) représentant la structure de la grille. Chaque feuille de l'arbre est associée à un *resource unit* et contient les données d'état du niveau correspondant et la référence du service associé. Cette arbre des données représente une vue fidèle de la topologie de la grille. Lorsque la défaillance d'un nœud de la grille est détectée, la feuille de l'arbre associée est supprimée.

Les données d'état collectées sur les machines font face à deux types d'utilisation : soit elles sont observées par un utilisateur, soit elles sont utilisées par les services eux-mêmes afin de réaliser un placement. Les informations d'état stockées par chaque feuille de l'arbre doivent permettre de répondre, le plus efficacement possible, aux contraintes liées à ces deux utilisations.

Les informations stockées à chaque niveau sont les suivantes :

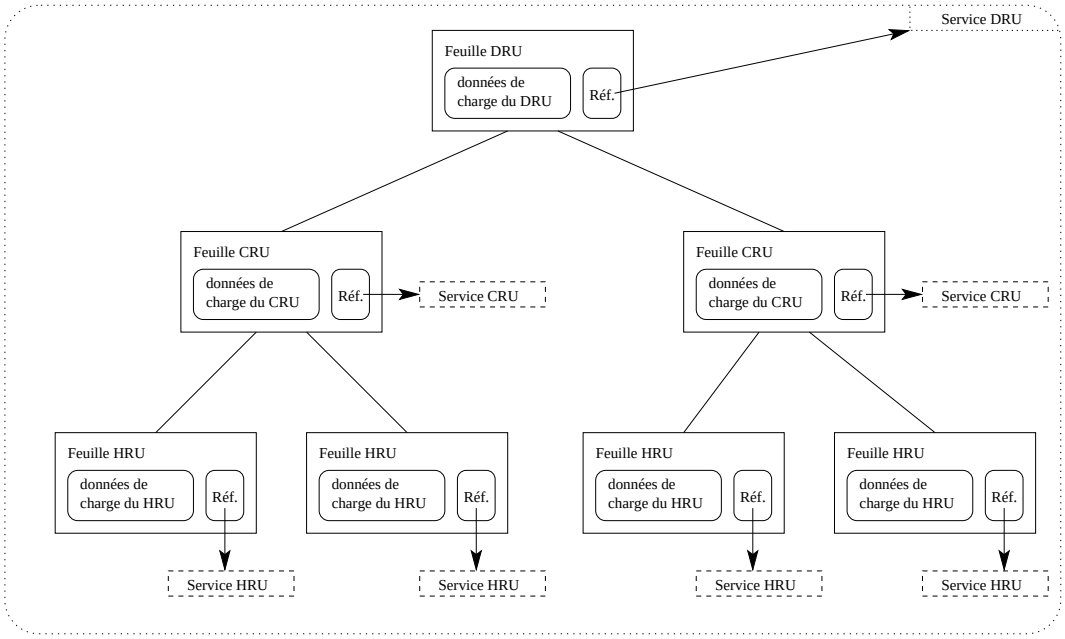


Fig. 4.6 – Données d'état stockées au niveau d'un domaine

Feuille HRU : charge de la machine sur les 6 dernières secondes (valeur paramétrable).

Feuille CRU : charge de chaque machine fille sur les 30 dernières secondes (valeur paramétrable) + charge de la grappe calculée en faisant la moyenne de la charge des fils. La présence de la charge de chaque nœud HRU est utile pour la prise d'une décision de placement.

Feuille DRU : charge de chaque machine fille sur la dernière minute (valeur paramétrable) + charge de chaque grappe fille sur la dernière minute + charge du domaine calculée en faisant la moyenne de la charge des grappes. La présence de la charge de chaque nœud HRU et CRU est utile pour la prise d'une décision de placement.

Feuille GRU : charge de chaque domaine sur les 15 dernières minutes (valeur paramétrable) + charge de la grille calculée en faisant la moyenne de la charge des domaines.

Un utilisateur souhaitant visualiser l'état de la grille ou d'une sous partie de la grille procède de la manière suivante :

1. Il se connecte à la racine de l'arbre ou du sous arbre dont il souhaite visualiser l'état.
2. Il parcourt l'arbre jusqu'aux feuilles et construit une représentation locale de l'arborescence découverte.
3. Il utilise les références des services fournies par chaque feuille de l'arbre afin de contacter directement chaque service observé. Cette dernière opération garantit l'obtention des informations les plus récentes et ainsi une vue la plus fidèle possible de l'état de la grille.

Lors d'une opération de placement, l'ordonnanceur utilise les informations d'état locales fournies par son module architecture afin de choisir les machines les plus appropriées. La vision de l'état des machines utilisée est moins fidèle à la réalité, mais on évite le sur-coût des communications liées au parcours de l'arborescence et au rapatriement des informations d'état les plus récentes.

4.5.3 Le module «ordonnanceur».

L'ordonnanceur d'Aroma permet de placer des tâches parallèles ou indépendantes tout en garantissant un certain niveau de qualité de service. Nous ne traitons pas ici de l'algorithme de placement utilisé, nous renvoyons pour cela le lecteur à la thèse de Mme Patricia PASCAL [58]. Nous détaillons toutefois le rôle de l'ordonnanceur et ses interactions avec les autres modules d'Aroma.

L'ordonnanceur utilise les données d'état, statiques et dynamiques, stockées dans le module architecture afin de choisir les machines les plus adaptées à l'exécution du traitement qui lui est confié. Il envoie ensuite un ordre d'exécution aux modules «lanceur» de chaque HRU sélectionné.

L'ordonnanceur conserve un état de l'application (liste des tâches, ressources consommées) qui est mis à jour régulièrement par des informations envoyées par le module «lanceur» de chaque HRU. Cette information est utilisée afin de tuer l'application en cours de traitement, de visualiser son état d'avancement ou de récupérer ses résultats.

Différentes décisions de placement peuvent être prises en fonction du niveau hiérarchique concerné :

- Niveau grille : l'ordonnanceur de niveau grille délègue la requête de placement au domaine le plus approprié. Les communications entre les tâches sont prises en considération afin d'orienter les tâches communiquant le plus vers le même domaine.
- Niveau domaine : l'ordonnanceur de niveau domaine peut, soit placer directement une tâche sur une des machines du domaine, soit déléguer la requête à une de ses grappes.
- Niveau grappe : l'ordonnanceur de niveau grappe place directement les tâches sur ses machines.

4.5.4 Le module «lanceur».

Le «lanceur» gère l'exécution des tâches sur chaque machine hébergeant un HRU. Ceci comprend :

- le lancement des tâches,
- le suivi des ressources consommées par chaque tâche,
- l'arrêt des tâches,
- la gestion des données d'entrée et des résultats.

4.6 Chargement dynamique de capteurs de charge.

Pouvoir modifier la liste des informations collectées sur les machines de la grille, ou modifier la manière dont certaines informations sont collectées, sans être obliger de stopper le fonctionnement de la grille est une fonctionnalité intéressante. En effet, arrêter et redéployer le gestionnaire de ressources sur un grand nombre de machines, est une opération fastidieuse et représenterait une rupture de service vis à vis du client. Des mécanismes ont donc été mis

en place afin de permettre d'ajouter aisément des capteurs de charge sur un ensemble de machines, sans nécessiter de rupture du service.

4.6.1 Principe utilisé

Nous utilisons ici la possibilité de chargement dynamique de classes offerte par le langage Java. Chaque serveur de la grille utilise des fichiers de configuration dans lesquels sont décrites les ressources à observer.

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

Les fichiers de configuration sont écrits en langage XML (cf. annexe A). Plusieurs outils existent pour récupérer les données contenues dans de tels fichiers. Nous utilisons JAXB[4] (*Java Architecture for XML Binding*), développé par *Sun Microsystems*, qui permet de récupérer le contenu de fichiers XML tout en vérifiant qu'ils respectent un schéma donné, et de le stocker sous forme d'objets Java.

Un fichier contient la liste des ressources à observer. Par la suite, un fichier par type de ressource fournit les informations utiles à la collecte de l'information :

- le nom de la ressource.
- le nom de la classe principale implémentant le capteur.
- le nom de l'archive JAR regroupant les classes définissant le capteur.

L'ajout d'un nouveau capteur est simplement réalisé en plaçant les classes implémentant le capteur dans une nouvelle archive, et en créant le fichier de configuration associé. Le chargement du capteur est alors réalisé de la manière suivante :

1. Le gestionnaire de ressources lit les fichiers de configuration et récupère les ressources.
   (a) lecture du fichier contenant la liste des ressources.
   (b) pour chaque ressource, lecture de son fichier de configuration spécifique, et création d'un objet contenant les informations contenu dans ce fichier.
   (c) les différents objets associés à chaque ressource sont transmis au module «collecteur».
2. Instanciation des capteurs de charge. Le module «collecteur» parcourt la liste des ressources à observer, et pour chaque ressource encore inconnue, le capteur est chargé en suivant les étapes suivantes :
   (a) l'archive JAR contenant la définition du capteur est récupérer et chargée dans la machine virtuelle Java en utilisant un `URLClassLoader` .
   (b) La classe principale du capteur est instanciée. Un «thread» exécutant la prise des mesures à intervalles réguliers est alors démarré.

[4] `http://java.sun.com/xml/jaxb/`

L'ajout d'un capteur de charge a également des répercutions au niveau de l'observateur de ressources et du module de statistiques.

4.6.2 L'observateur de ressources

Il s'agit d'un module permettant à un client de visualiser l'état des différentes ressources des machines sur lesquelles il est connecté. Dans une interface graphique, l'utilisateur voit les ressources disponibles, sélectionne celles qui l'intéressent, puis lance une observation. Les valeurs mesurées sur les machines sont alors affichées sous forme de graphiques en pseudo temps réel (rafraîchissement périodique de quelques secondes).

L'interface graphique doit pouvoir s'adapter aux capteurs disponibles sur chacune des machines auxquelles elle est connectée. Pour cela, l'observateur de ressources demande aux serveurs auxquels il est connecté de lui fournir la liste des capteurs de charge disponibles. Ainsi, les ressources disponibles au niveau de l'observateur correspondront parfaitement aux ressources observables sur chacun des serveurs.

4.6.3 Le module de statistiques

Ce module est responsable de l'archivage des valeurs des différentes ressources dans une base de données. Chaque serveur envoie périodiquement les valeurs relevées sur sa machine par les différents capteurs de charge à ce module. Toutes ces données peuvent ensuite être consultées à partir de l'interface cliente.

Les ressources observables ne sont pas connues à l'avance. Elles doivent au contraire pouvoir être ajoutées en cours d'exécution tout en étant assurées que leurs valeurs soient tout de même sauvegardées dans la base de données.

Pour résoudre le problème de sauvegarde des nouvelles ressources dans la base de données, à la réception d'une nouvelle série d'observations en provenance d'un serveur, le module qui s'interface avec la base de données crée une requête SQL demandant l'insertion d'une ligne contenant toutes les valeurs observées. Avant de soumettre la requête, ce module vérifie que toutes les colonnes nécessaires existent bien. Si ce n'est pas le cas, il soumet alors une requête de création de colonnes.

4.7 Tolérance aux pannes : utilisation de réplicas

Dans un environnement impliquant un nombre important de machines tel que les grilles, la défaillance d'un équipement est un événement naturel, dont le déclenchement ne doit pas remettre en cause le bon fonctionnement de la structure. Le type de pannes pouvant intervenir, leur impact sur le comportement du gestionnaire et les solutions apportées aux problèmes potentiels vont maintenant être présentés.

4.7.1 Les différents types de pannes possibles.

Notre objectif n'est pas de mener une étude détaillée de l'ensemble des défaillances logicielles ou matérielles, pouvant intervenir dans le cadre de la gestion des ressources de la grille, et des répercutions qu'auraient ces défaillances sur le comportement d'Aroma, mais simplement de proposer des mécanismes permettant de faire face à deux type de pannes simples : l'arrêt complet d'un service et les coupures réseau.

4.7.1.1 L'arrêt d'un service

Ce type de panne est le plus évident à détecter. Il s'agit de l'arrêt complet d'un service pour une raison quelconque. L'arrêt du logiciel Aroma sur une des machines, du fait d'un bug par exemple, ou bien encore l'arrêt complet de la machine, du fait d'une panne matérielle, sont deux exemples des nombreuses causes pouvant entraîner ce type de pannes.

L'arrêt du logiciel Aroma sur la machine concernée peut entraîner de ce fait une perte d'information, par exemple les tâches en cours d'exécution sur la machine seront perdues.

4.7.1.2 Les coupures réseau

Nous considérons ici, uniquement les pannes d'équipements réseau pouvant aboutir à la rupture totale des communications entre deux machines. Ce type de pannes implique la rupture des communications entre différentes parties de l'architecture Aroma.

La difficulté pour gérer ce type de pannes est due au fait qu'aucun service n'est arrêté. Il faut donc que chaque service détecte les coupures du réseau, et surtout détecte le rétablissement de ce dernier, afin de poursuivre éventuellement le dialogue.

4.7.2 Impact des défaillances sur le comportement du gestionnaire de ressources Aroma.

4.7.2.1 Arrêt d'un service

La perte d'un service revêt une importance différente selon le niveau hiérarchique à laquelle elle intervient.

- niveau machine : la défaillance d'un service HRU implique la perte des tâches s'exécutant sur la machine associée, et la perte d'une ressource de calcul. Un tel événement ne met cependant pas en péril le fonctionnement des autres composants de la grille.
- niveau grille : la défaillance d'un service GRU implique la perte d'un point d'accès à la grille. De la même manière, l'impact d'un tel événement est mineur sur le comportement de la grille.
- niveau grappe et domaine : les services CRU et DRU représentent des nœuds intermédiaires de l'architecture hiérarchique du gestionnaire de ressources. De ce fait, la perte d'un de ces nœuds peut avoir des conséquences importantes sur le fonctionnement du gestionnaire. Un tel événement entraîne la perte de l'ensemble des informations concernant les applications placées par le nœud défaillant. Ainsi, les applications continueront leur exécution sur les nœuds HRU, mais il sera impossible de suivre cette exécution ou d'en récupérer les résultats. De plus, notamment dans le cas de la perte d'un CRU, toutes les machines appartenant à la grappe associée deviendront inaccessibles via le gestionnaire de ressources.

4.7.2.2 Coupures réseau

Les coupures réseau posent des problèmes de cohérence de la vision de l'architecture de la grille, et peuvent entraîner le partitionnement de la grille en sous arbres indépendants incapables de dialoguer entre eux. En effet, un service CRU ne réussissant plus à dialoguer avec un de ses HRU fils, va décider de supprimer ce fils, le considérant comme «mort». Cependant, le service HRU continue d'exister et notamment poursuit la supervision de l'exécution des tâches qui lui ont été confiées. Il pourra également, par la suite transmettre les résultats de

ces tâches à son CRU père si le réseau est rétabli. Le service père risque d'ignorer les résultats reçus car pour lui son fils est mort. Les même problèmes peuvent intervenir à chaque niveau hiérarchique, impliquant la perte de parties entières de la grille.

4.7.3 Les stratégies mises en place

Un mécanisme de réplication passive [76] est utilisé pour traiter cette problématique. Le principe des services actifs et réplicas[6], utilisé afin de pallier les défaillances décrites précédemment va maintenant être exposé.

4.7.3.1 Services actifs et services réplicas

Le principe des services actifs et réplicas est très simple. Il repose sur une notion très répandue dans les systèmes de tolérance aux fautes : la redondance. Il s'agit, en effet, de doubler certains services critiques, afin que, si un service est momentanément indisponible, un autre service similaire puisse prendre le relais, et ainsi permettre au système Aroma de conserver une architecture cohérente et fonctionnelle. Le service répondant aux requêtes est appelé service actif, l'autre service répliqua.

Seuls les DRU et les CRU peuvent être secondés par des réplicas, c'est-à-dire les services se retrouvant sur les nœuds intermédiaires de l'architecture d'un système Aroma.

Chaque service actif de type DRU et CRU ne possède au maximum qu'un service réplica de même type. Ceci permet de s'affranchir des difficultés liées au choix du réplica devant prendre le relais d'un service actif lorsque celui-ci n'est plus disponible. Cependant, ceci limite le niveau de tolérance aux pannes.

4.7.3.2 Problématique

Chaque service réplica doit être en mesure de détecter l'arrêt du service actif qu'il seconde, puis de prendre le relais, c'est-à-dire devenir actif et s'insérer correctement dans l'architecture. Il est évident que les services actifs et réplicas doivent disposer des mêmes informations et être en permanence dans le même état (afin que, si le service actif s'arrête, le réplica n'ait perdu aucune information).

Ensuite, il est nécessaire de s'assurer, qu'à tout instant, il n'y ait qu'un service actif de même type et de même nom. Ainsi, si deux services actifs similaires sont présents à la fois, l'un d'eux doit, de lui-même, devenir réplica.

Chaque service actif doit connaître tous ses parents, que ces derniers soient actifs ou bien réplicas. Une communication sera établie entre enfants et parents, afin que les services enfants puissent envoyer des informations sur leur état à leurs parents. Ces informations seront indépendantes vis-à-vis du rôle joué par le parent les recevant, c'est-à-dire que ce dernier soit actif ou bien réplica.

Il est à noter qu'un service réplica ne peut avoir de parents, de manière à ne pas dédoubler l'arbre de l'architecture. En effet, si une branche était établie entre le DRU réplica et le GRU au même titre qu'entre le DRU actif et le GRU, l'exploration de l'arbre de la racine (GRU) vers les feuilles (HRU), en prenant l'exemple d'un DRU actif secondé par un DRU réplica, amènerait le système à croire que deux sous-arbres seraient présents sous le GRU, alors qu'il n'en est rien.

Enfin, il est à noter qu'une communication est établie entre chaque service actif et son réplica associé, afin de faire passer des informations de mise à jour de l'un vers l'autre. Ce point est étudié plus en détail dans la dernière partie (section 4.7.4)

La figure 4.7 montre une architecture Aroma utilisant le système de tolérance aux fautes.

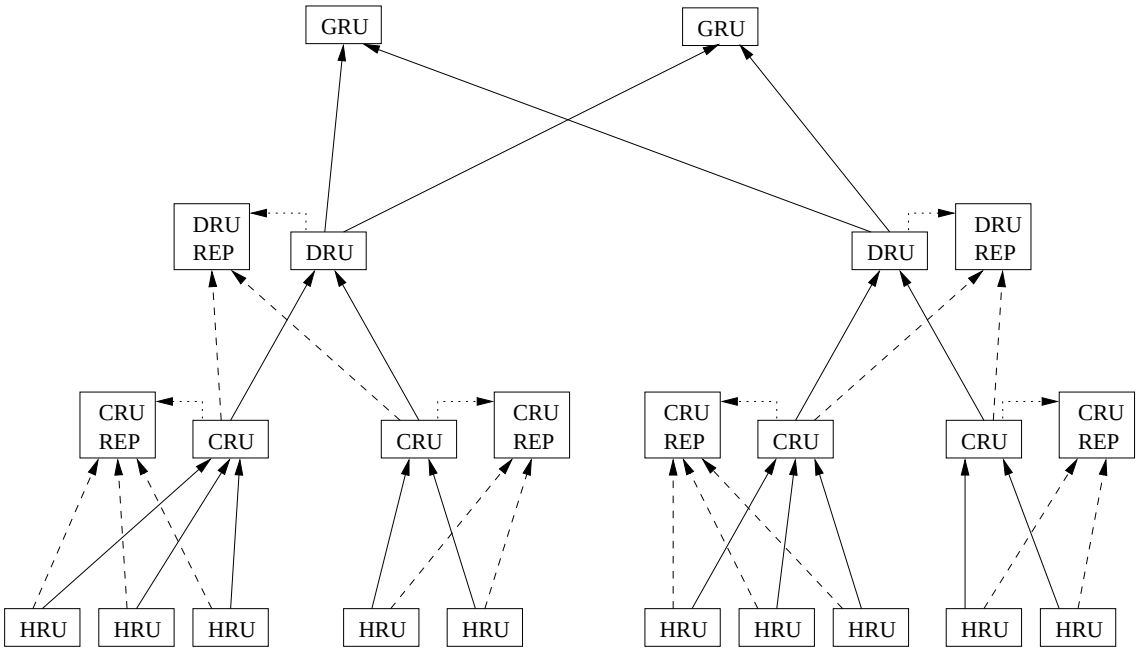


Fig. 4.7 – Exemple d'architecture avec la mise en place du système de tolérance aux fautes

4.7.3.3 Démarrage des services Aroma

Deux types distincts de services sont présents : les services actifs et les services réplicas. La phase de lancement de chacun de ces deux types de services va être décrite.

Lancement de services actifs

Cette partie concerne les services actifs, qui peuvent être de type DRU ou CRU.

La première opération à effectuer, dans le cas où le service est un DRU ou un CRU, est de vérifier qu'il n'existe pas déjà un service actif équivalent (même nom, même type), ceci afin d'éviter le cas où deux services actifs similaires évolueraient en même temps. Pour cela, les *lookups services* sont interrogés, et le programme bloque son exécution jusqu'à ce qu'un *lookup service* lui réponde. Si aucun *lookup service* ne répond dans un délai de deux minutes, le programme continue son exécution. Si le *lookup service* trouve un service actif, et renvoie sa référence, le service en cours de démarrage devient réplica. Si aucun service n'est retourné, le service en cours de démarrage peut (et doit) rester actif. L'exécution du programme peut ensuite continuer.

Dans un second temps, la phase de découverte d'un *DataManager* intervient. Le *DataManager* est un service dont le rôle est de gérer la base de données statistique d'Aroma.

La phase suivante concerne la recherche éventuelle de parents actifs et de parents réplicas auprès des *lookup services associés*. Selon le type du service concerné, cette opération différera légèrement :

Service DRU : Un maximum de parents actifs est recherché.
Service CRU : Un parent actif et un parent réplica seulement sont recherchés.

Lorsque cette phase a été complétée, le service est opérationnel et intégré au sein de l'architecture hiérarchique d'Aroma, il s'enregistre auprès des *lookup services* de son groupe afin de pouvoir être contacté par un client ou un de ses fils.

Lancement de services réplicas

La phase de mise en place d'un service réplica va maintenant être décrite. Elle est bien plus simple que celle d'un service actif.

La première opération effectuée ici est la recherche d'un *DataManager*, identique à celle mentionnée auparavant.

Ensuite, le service réplica s'enregistre auprès des *lookup services* de son groupe.

Enfin, le service réplica recherche le service actif qu'il devra surveiller. Il est à noter que cette recherche doit être bloquante, pour éviter le cas où, lors du démarrage d'Aroma, le service réplica, pouvant être lancé avant le service actif, ne détecterait aucun service actif (puisque ce dernier n'aurait pas encore été créé), et donc deviendrait actif. Cependant, si après une durée de cinq minutes aucune présence de service n'est détectée, le service réplica devient actif.

Il est à noter que la phase de recherche des parents est ici inexistante, puisque, comme cela a été mentionné précédemment , un service réplica ne peut, en aucun cas, avoir de parents.

Par la suite, nous illustreront nos propos en nous basant sur l'architecture type représentée par la figure 4.8.

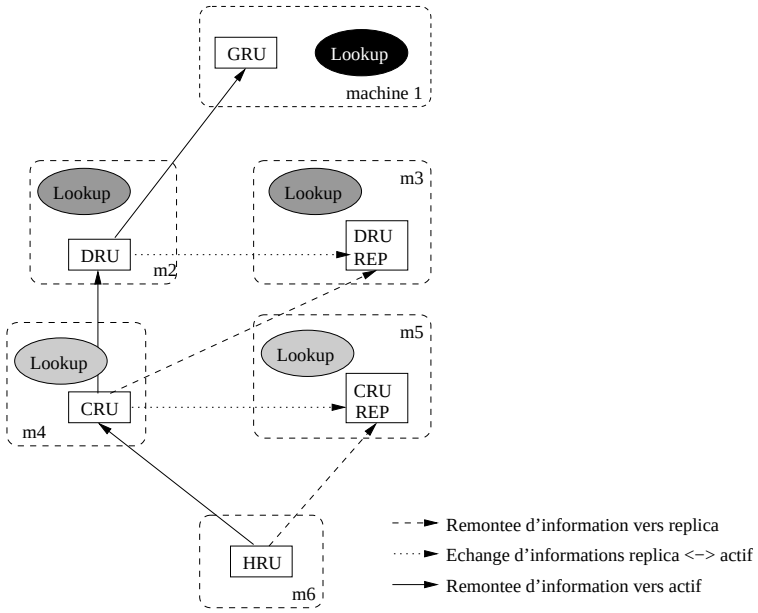


Fig. 4.8 – Exemple d'arbre décrivant une architecture Aroma possible

4.7.3.4 Arrêt d'un service

Avant de songer aux mécanismes à mettre en place afin de pallier l'arrêt d'un service, il est nécessaire d'être capable de détecter un tel événement. Ceci peut être simplement réalisé grâce

aux messages transmis entre les différents services (voir section 4.4.3). Comme nous l'avons expliqué précédemment, chaque message est associé à un *lease* dont l'expiration permet de détecter la rupture d'une communication entre deux services. Il n'est cependant pas possible d'identifier si la rupture de la communication est due à la défaillance d'un nœud ou à des problèmes réseau. Le non-renouvellement d'un *lease* déclenche tous les mécanismes de gestion de l'arrêt d'un service.

Deux types de *leases* différents ne sont pas renouvelés en cas d'arrêt d'un service : le *lease* associé à la communication entre le service actif et le service réplica et les *leases* associés aux communications entre chaque service fils et le service père (ce dernier cas peut lui-même être décomposé en deux sous-cas : lorsque le parent est actif, et lorsqu'il est réplica).

Arrêt d'un service réplica

Lorsque le service qui s'arrête est un service réplica, le mécanisme de *lease* décrit précédemment, permet au service actif qu'il était sensé surveiller ainsi qu'à ses enfants de détecter sa mort.

Le service actif n'a aucun traitement particulier à réaliser, si ce n'est autoriser un autre réplica à lui demander des informations. Côté enfant, lorsque l'arrêt du service parent réplica est détecté, une recherche auprès des *lookup services* est lancée afin de détecter un autre parent réplica.

Dans l'exemple décrit précédemment, l'arrêt du CRU réplica aboutirait à l'architecture de la figure 4.9 :

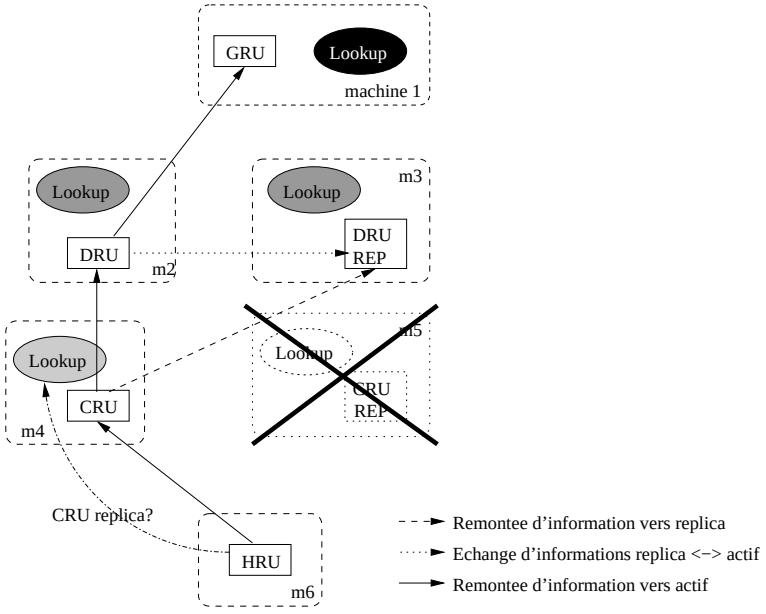


Fig. 4.9 – Arbre décrivant l'exemple d'architecture Aroma après l'arrêt du service CRU réplica

Arrêt d'un service actif

Dans le cas où un service actif s'arrête, son réplica, s'il existe, doit prendre le relais et devenir actif. Les enfants de ces services doivent prendre en compte ce changement : leur parent réplica doit devenir leur parent actif, et ils doivent ainsi rechercher un nouveau parent réplica.

La prise de décision est répartie entre les différents services. Lorsque le service réplica détecte la mort du service actif qu'il surveille, il décide de devenir actif. Pour cela, il contacte chaque *lookup service* auprès duquel il est déclaré afin de modifier son statut de l'état de réplica à l'état d'actif. Il contacte ensuite ses enfants afin de leur signifier qu'il cesse d'être leur parent réplica. Du point de vue des enfants, deux événements sont alors détectés (dans un ordre quelconque) : la mort de leur parent actif (il leur faudra alors chercher un autre parent actif auprès des *lookup services*) et l'abandon de leur parent réplica (il leur faudra alors rechercher un autre parent réplica auprès de ces mêmes *lookup services*).

Plusieurs pôles de décision sont présents : un au niveau du réplica qui décide de passer en actif et d'abandonner ses enfants, et un autre au niveau de chacun des enfants qui décident de rechercher un parent actif et un parent réplica.

L'avantage majeur de cette stratégie par rapport à une décision centralisée, du fait de la dissociation totale au niveau des enfants entre leur parent actif et leur parent réplica, est que l'ordre dans lequel la détection de l'arrêt du service actif se fait (entre son réplica et ses enfants) n'aura aucune influence sur le programme.

Le seul inconvénient de cette méthode vient du fait que les *lookup services* sont interrogés pour obtenir une référence de service qui est déjà connue (puisqu'il s'agit de celle de l'ancien parent réplica qui est devenu actif).

Le service réplica nouvellement devenu actif doit s'enregistrer auprès de ses parents afin de conserver une architecture cohérent. Pour cela il interroge les *lookup services* afin d'obtenir la référence de ses parents (actifs, et éventuellement réplica), et de les contacter. Les parents détecterons la mort d'un de leur fils, et le remplaceront par le réplica lorsque celui-ci les contactera.

Il est utile de rappeler que, lorsque le service actif défaillant sera relancé, il détectera la présence d'un service actif le remplaçant, et deviendra de lui-même réplica, afin de garantir la présence d'un seul service actif au sein de l'architecture.

Dans l'exemple décrit précédemment, quelle que soit la stratégie employée, l'arrêt du service CRU actif aboutirait à l'architecture de la figure 4.10 :

4.7.3.5 Coupures réseau

La détection d'une coupure réseau est effectuée de la même manière que la perte d'une machine, grâce à l'utilisation des *leases*. Cependant, le fait que les deux services communicants soient toujours fonctionnels, et donc capables de détecter l'arrêt de la communication est une difficulté supplémentaire à gérer. En effet, chaque entité doit prendre localement une décision mais il ne faut pas que deux entités prennent des décisions contradictoires.

Un exemple de problème pouvant survenir est le suivant : imaginons que la liaison entre le service actif et son réplica associé soit momentanément rompue, le service réplica, croyant détecter l'arrêt du service actif, va devenir actif. Pendant ce temps, le service actif initial continuera son exécution. Par conséquent, lorsque le réseau sera rétabli, deux services actifs cohabiteront, ce qui n'est bien sûr pas tolérable. Le seul moyen est de faire en sorte que l'un de ces deux services devienne réplica.

Il est à noter que de nombreux éléments mis en place pour gérer l'arrêt de services sont réutilisés pour gérer les coupures réseau, en y apportant cependant quelques modifications. Ces modifications ont pour but de faire en sorte qu'il n'y ait jamais deux services actifs présents simultanément.

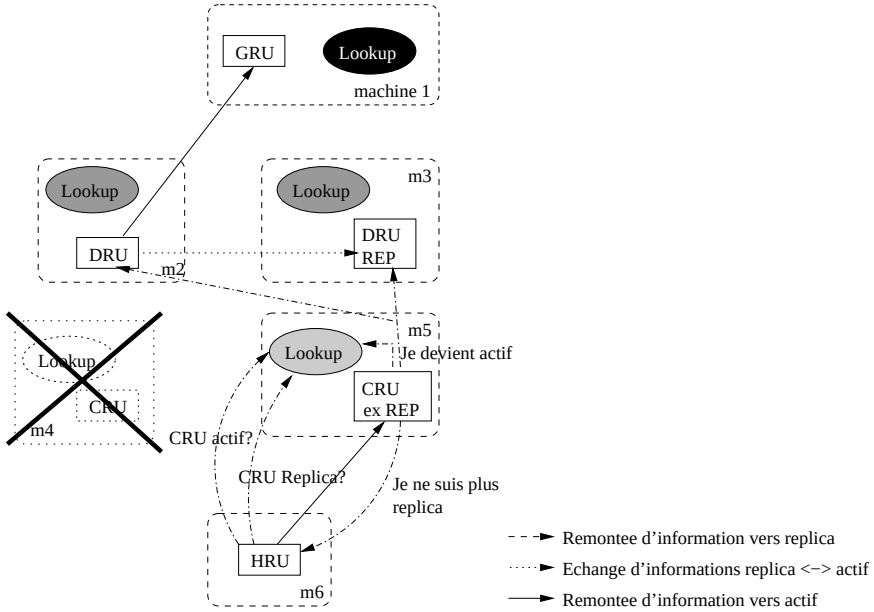


Fig. 4.10 – Arbre décrivant l'exemple d'architecture Aroma après l'arrêt du service CRU actif

La stratégie mise en place modifie uniquement le comportement du service actif initial. Ainsi, lorsqu'une coupure réseau intervient, en supposant qu'elle implique une rupture de la liaison entre le service actif et son réplica associé, le service réplica devient actif, comme il l'aurait fait si le service actif s'était arrêté.

Comme ce service devient actif, il autorise n'importe quel réplica à venir le surveiller. De son côté, le service actif initial reste actif. Seulement, il demande au *lookup services* de le prévenir lorsqu'un service actif (autre que lui-même) viendra s'enregistrer. Ceci sera effectué lorsque le réseau sera rétabli. Le *lookup service* donnera au service initialement actif la référence du service réplica devenu actif. Ainsi, le premier de ces services détectera la présence d'un autre service actif, et par conséquent deviendra réplica de cet actif.

Le pouvoir de décision est réparti entre les différents services constituant l'architecture. Ainsi, lorsqu'un service actif devient réplica, il avertit simplement ses enfants qu'il n'est plus leur parent actif. Ces derniers n'auront plus qu'à lancer une recherche pour retrouver leurs nouveaux parents.

4.7.4 Communication entre service actif et service réplica

Cette partie concerne l'échange d'informations entre un service actif et le service réplica qui le surveille. Ce mécanisme complète le système de tolérance aux fautes. En effet, seuls les mécanismes permettant d'avoir, à chaque instant, une architecture correcte et cohérente, ont été abordés. Comme cela a été exprimé précédemment, ces mécanismes reposent sur les communications établies entre chaque service.

Cependant, le contenu de ces communications n'a pas été, jusqu'à maintenant, explicité.

Le rôle des services actifs est de placer des tâches sur les services HRU. Ces derniers transmettent périodiquement des informations sur leur état à leurs parents, qu'ils soient actifs ou bien réplicas.

Ainsi, lorsqu'un CRU actif reçoit des informations en provenance de ses HRU fils, il peut

les traiter puisqu'il a connaissance de chacune des tâches qui ont été lancées sur chacune des machines abritant les HRU. Cependant, le service CRU réplica qui recevra ces mêmes informations ne pourra pas les traiter, puisqu'il n'a aucune indication quant aux tâches lancées sur les HRU.

Le but des communications entre service actif et service réplica est d'amener le service réplica à avoir exactement les mêmes connaissances que le service actif qu'il surveille. Ainsi, si le service actif devient indisponible, le service réplica pourra prendre le relais sans aucun problème.

Chaque service actif de type DRU ou bien CRU envoie périodiquement des messages au réplica qui le surveille. Chacun de ces messages contient la liste des applications ordonnancées et la liste des applications en cours d'ordonnancement. Le service réplica connaît ainsi les noms des applications, les tâches et la machine sur laquelle elles ont été placées.

4.8 Conclusion

L'architecture du gestionnaire de ressources Aroma et ses principales fonctionnalités ont été présentées. La caractéristique principale de cette architecture est son organisation en niveaux hiérarchiques qui facilite le passage à l'échelle, en agrégeant les informations conservées à chaque niveau. Une originalité de ce gestionnaire est également de pouvoir prendre une décision de placement de tâches à différents niveaux hiérarchique (niveau grappe, domaine, grille). Cette caractéristique permet par exemple, d'utiliser en priorité ses ressources locales (niveau grappe) tout en ayant la possibilité d'accéder à des ressources supplémentaires en cas de pics d'activité (niveau domaine).

Aroma se distingue des autres *Network Enabled Servers* par l'utilisation de l'intergiciel de communication Jini et par l'utilisation de capteurs de charges propriétaires. Les concepts de base de Jini sont utilisés afin de faciliter la communication entre les services. Ils permettent de s'affranchir de la programmation des couches basses du réseau et offrent des solutions intéressantes utilisées, notamment, pour la détection des pannes. L'utilisation de capteurs de charge propriétaires permet de minimiser le coût de la collecte d'information (en terme de consommation de ressources) et également de choisir en détail les informations collectées. Cette dernière caractéristique est importante, car de la précision des informations collectées dépend la qualité des décisions de placement prises.

Chapitre 5

Spécificités de la mise en ASP d'Aroma

Sommaire

5.1 Introduction

L'utilisation du gestionnaire de ressources dans un mode ASP (*Application Service Provider*) figure 5.1 soulève de nouveaux problèmes. Notre étude se positionne dans un contexte mettant en jeu deux acteurs principaux : un client désirant exécuter un calcul complexe et un fournisseur de services mettant à disposition ses machines et ses applications. Dans ce contexte d'utilisation, un contrat passé entre chaque client et le fournisseur de services, doit permettre de définir les attentes du client et les modalités de facturation du service vis à vis de ce client.

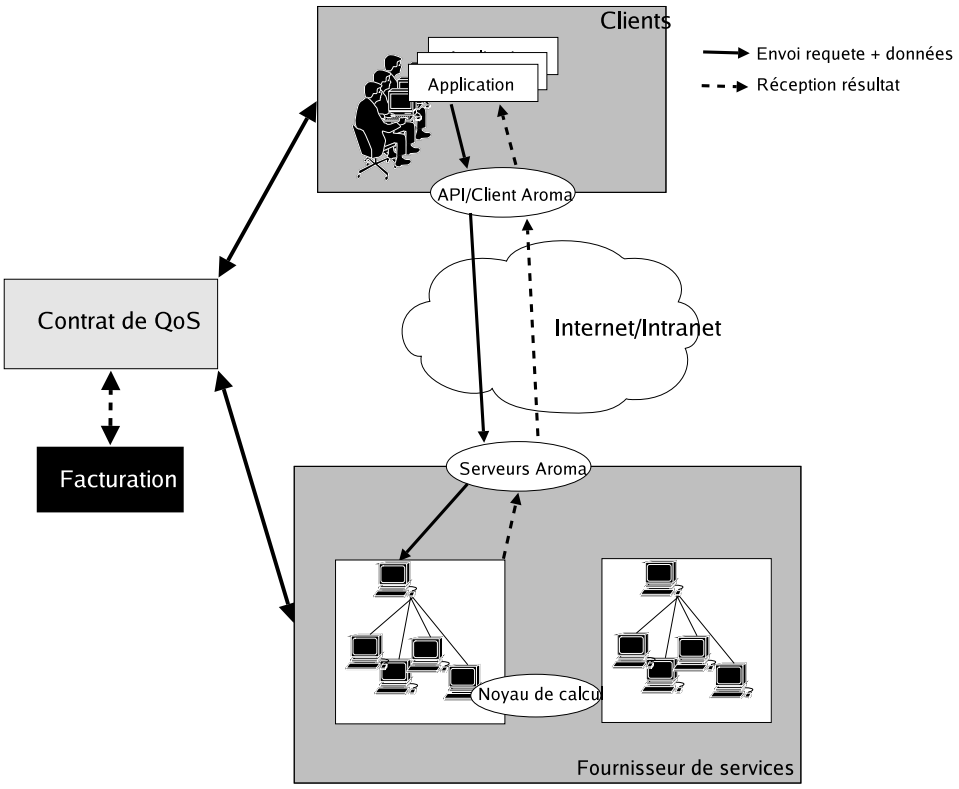


Fig. 5.1 – Utilisation d'Aroma en mode ASP

La confidentialité des données échangées, des codes de calcul et l'authentification des clients sont également des problèmes à prendre en considération. De même, la facilité de portage d'applications existantes, en mode d'utilisation ASP, est un aspect à ne pas négliger afin de ne pas rebuter les utilisateurs potentiels.

5.2 La notion de contrat

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

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

5.2.1 Groupe d'utilisateurs

Définir un contrat particulier pour chaque utilisateur de la grille serait une opération fastidieuse et inutile. Afin d'éviter cela, Aroma permet d'associer un ensemble d'utilisateurs à un même contrat [72].

De ce fait, le contrat peut représenter une équipe travaillant sur un projet commun, un service d'une entreprise ou la totalité d'une entreprise. Le contrat permet de définir le cadre global d'utilisation du service, chaque groupe d'utilisateur est ensuite libre de répartir les

ressources dont il dispose entre les différents membres du groupe. Pour ce faire, tout contrat possède un ou plusieurs utilisateurs particuliers chargés d'administrer le contrat.

On distingue ainsi deux niveaux d'administration d'un contrat. Chez le fournisseur de service, un administrateur global se charge de :

- définir les limites d'utilisation des ressources pour le contrat,
- définir le niveau de qualité de service associé au contrat,
- définir le niveau de tarification du service,
- créer au moins un utilisateur pour le contrat. Cet utilisateur doit également être administrateur du contrat.

Chez le client, un ou plusieurs utilisateurs possédant des droits d'administrations peuvent :

- répartir l'utilisation des ressources entre chaque utilisateur du contrat,
- créer de nouveaux utilisateurs pour son contrat.

5.2.2 Informations associées au contrat

Le contrat permet de définir des limites maximales et minimales sur l'utilisation des ressources, notamment :

- le temps processeur que peuvent consommer les utilisateurs du contrat,
- la quantité de mémoire que peuvent posséder les machines utilisables par les utilisateurs du contrat,
- le nombre de processeurs pouvant être utilisés pour traiter une même application du contrat,
- applicatifs utilisables via le contrat,
- etc.

Il permet également de définir le niveau de qualité de service associé au contrat. Ce niveau de qualité de service permet de définir les contraintes que peut exprimer un utilisateur lors de chaque requête de placement de tâches (application avec ressources dédiées, date butoir...).

Enfin, le mode de tarification associé à la prestation est également défini par le contrat. Cette tarification est essentiellement basée sur les ressources consommées et le niveau de qualité de service demandé. Des travaux sont actuellement en cours afin d'étudier différents modèles économiques et de les implémenter dans Aroma [5].

L'ensemble des informations associées au contrat sont stockées dans une base de données (figure 5.2) centralisée, qui est traitée par le *DataManager service*.

5.2.3 Traitement des informations du contrat

Chaque utilisateur du contrat reçoit, par défaut, les mêmes limites que celles associées à son contrat. Ces limites peuvent ensuite être restreintes par l'administrateur du contrat.

Lorsqu'un utilisateur formule une demande de placement, ce dernier indique le niveau de qualité de service qu'il souhaite voir associer à sa requête, les ressources nécessaires à l'exécution de sa requête (nombre de processeurs, quantité mémoire...), ainsi qu'une prédiction du temps processeur nécessaire à l'exécution de sa requête. Une vérification est effectuée, afin de garantir que les droits de l'utilisateur et de son contrat ne soient pas violés par la requête.

Durant l'exécution de l'application, les informations sur les ressources consommées par l'application, fournies par le module *lanceur*, sont utilisées afin de vérifier, d'une part, que l'application ne consomme pas plus de ressources que son contrat ne lui autorise, et d'autre part de stocker ces informations dans la base de données. La première vérification (figure 5.3)

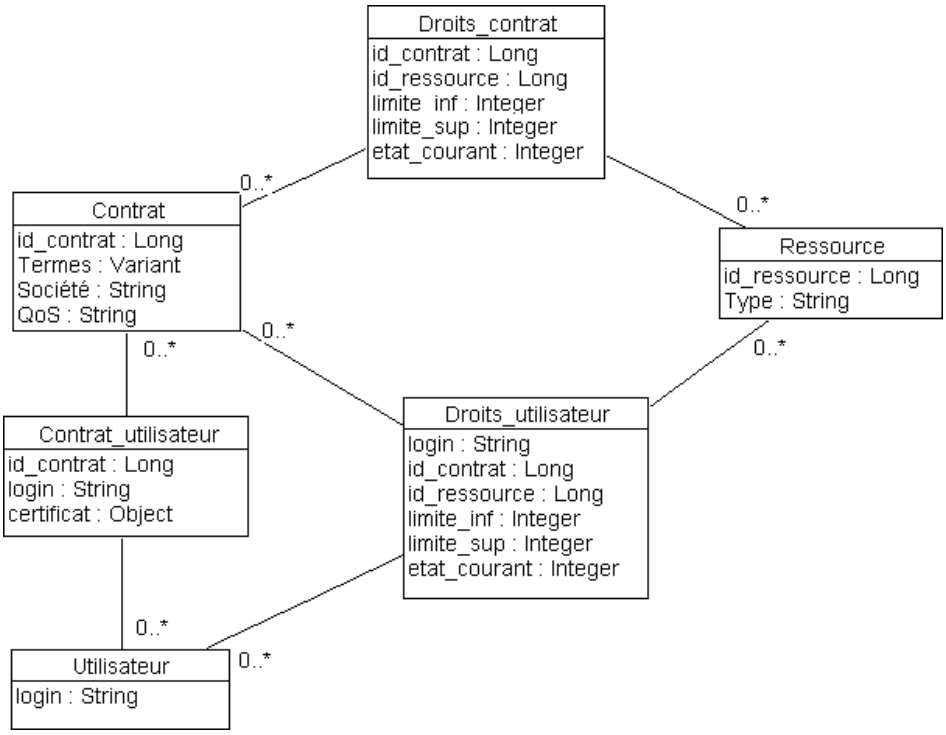


Fig. 5.2 – Modèle relationnel des utilisateurs d'Aroma

permet de traiter le cas où l'utilisateur fournit une prédiction erronée du temps processeur nécessaire à l'exécution de son traitement. Dès que les ressources consommées dépassent d'un certain pourcentage les ressources allouées à l'utilisateur, l'application peut être stoppée par le gestionnaire de ressources. La sauvegarde régulière des ressources consommées, dans la base de données, permet de conserver une trace de l'exécution même en cas de défaillance du noeud d'exécution.

5.3 La sécurité

La sécurité est un problème délicat à traiter. Elle concerne à la fois l'authentification des utilisateurs et des fournisseurs de services, le chiffrement des communications, la non-répudiation, et la sécurité des données et des codes de calcul [72].

5.3.1 Authentification des utilisateurs.

Aroma utilise une authentification des utilisateurs par certificat numérique *X509*. Chaque utilisateur du gestionnaire doit posséder un certificat numérique qui permet de l'authentifier. Lors d'une connexion au gestionnaire de ressources, le mot de passe du certificat doit être saisi afin de poursuivre tout dialogue avec les services. L'authentification repose sur l'utilisation de JAAS[1](Java Authorization and Authentication Service), ce qui permet de modifier facilement le type d'authentification utilisé (figure 5.4) en fonction du niveau de sécurité souhaité (simple mot de passe, kerberos...).

[1] `http://www.java.sun.com/JAAS`

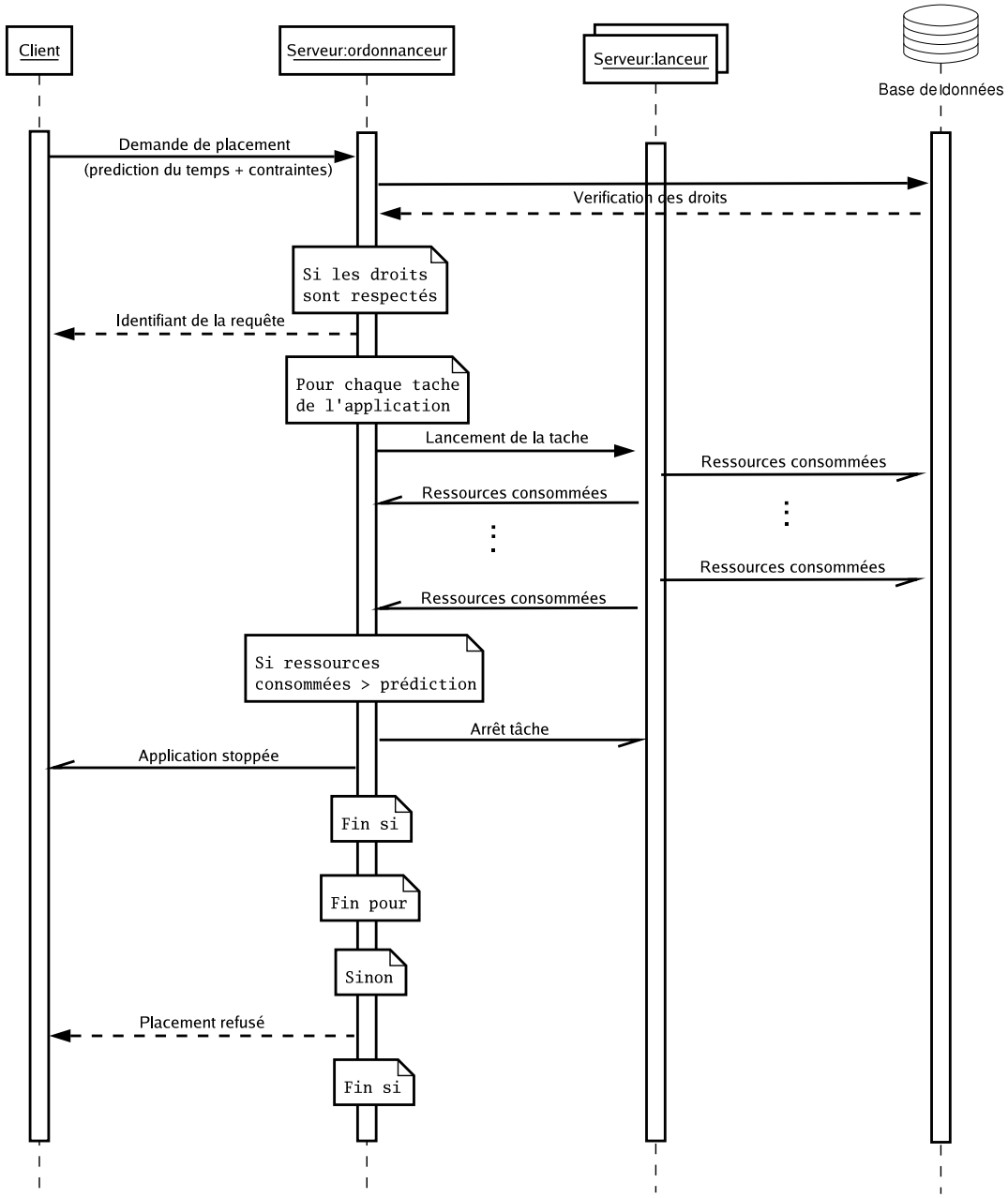


Fig. 5.3 – Dialogue avec la base de données lors d'une soumission de tâches

5.3.2 Chiffrement des communications

La version de Jini utilisée (v1.2) n'offre aucun mécanisme de sécurité. De ce fait, une solution partielle basée sur l'utilisation de JSSE[2] (Java Secure Socket Extension) et du protocole TLS (Transport Layer Service) a été mise en place.

Cette solution assure l'authentification mutuelle entre le client et le serveur, ainsi que le chiffrement des communications réalisé grâce aux clés publiques et privées de chaque prota-

[2] `http://www.java.sun.com/JSSE`

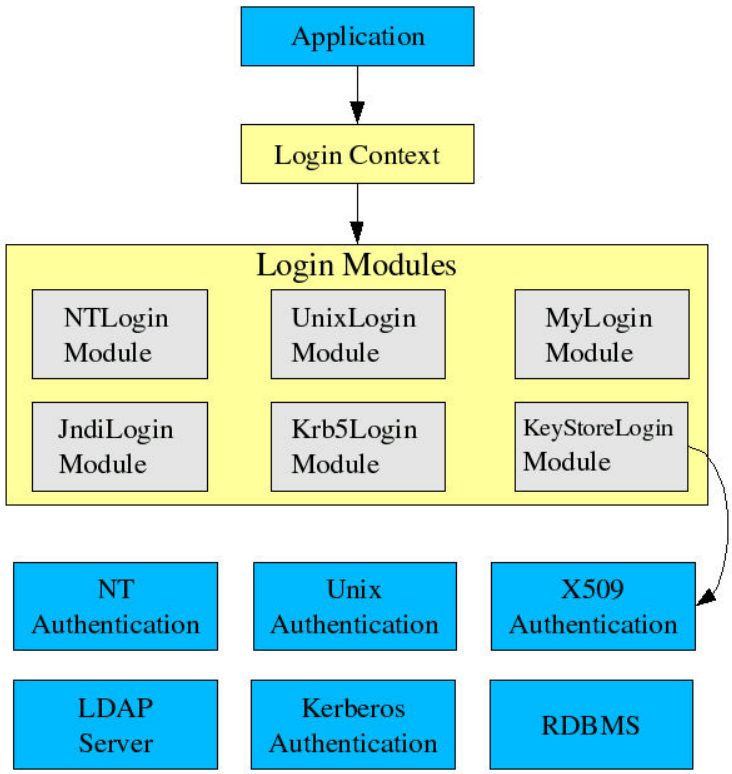


Fig. 5.4 – Architecture d'authentification de JAAS

goniste.

Les dialogues avec les *lookup services*, au niveau des clients ou des serveurs, restent cependant non sécurisés. La dernière version de Jini (v2.0) offre une architecture de sécurité permettant de palier les faiblesses de la solution mise en place.

5.3.3 Protection des données et du code.

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

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

Ces mécanismes permettent d'assurer le même niveau de protection que sur une machine standard.

5.4 Interactions entre un client et le gestionnaire de services

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

Afin de répondre aux besoins de ces deux modes d'utilisation, il a été choisi d'utiliser un client léger pour se connecter au gestionnaire de ressources. L'interaction entre ce client léger et le gestionnaire de services, et tout particulièrement les aspects de ces dialogues visant l'authentification et l'autorisation des utilisateurs vont maintenant être décrits.

5.4.1 Le client Aroma : problématique et avantages

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

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

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

5.4.2 Vérification des droits via l'API

Lors d'un dialogue par l'intermédiaire de l'API (figure 5.5), la première étape concerne l'authentification du client grâce à son certificat numérique. Cette étape est un traitement local permettant de s'assurer que la personne physique souhaitant soumettre une requête est bien la personne identifiée par le certificat numérique (ou une personne connaissant le mot de passe associé au certificat numérique). L'identité de la personne authentifiée est conservée par le client léger durant la durée de la transaction, et est fournie lors de chaque communication avec le gestionnaire de ressources. A chaque requête du client, le gestionnaire de ressources interroge sa base de données, afin de déterminer si le client est autorisé à effectuer le traitement concerné. De cette manière, il est possible d'interdire à un client de se connecter à certaine partie de la grille, ou d'accéder à certaines fonctionnalités du gestionnaire.

5.4.3 Vérification des droits via l'interface graphique

Lors de l'utilisation de l'interface graphique d'Aroma, un certain nombre de vérifications sont également réalisées afin connaître les fonctionnalités du gestionnaire de ressources auxquelles l'utilisateur peut accéder. L'interface graphique est conçue de manière modulaire, ainsi

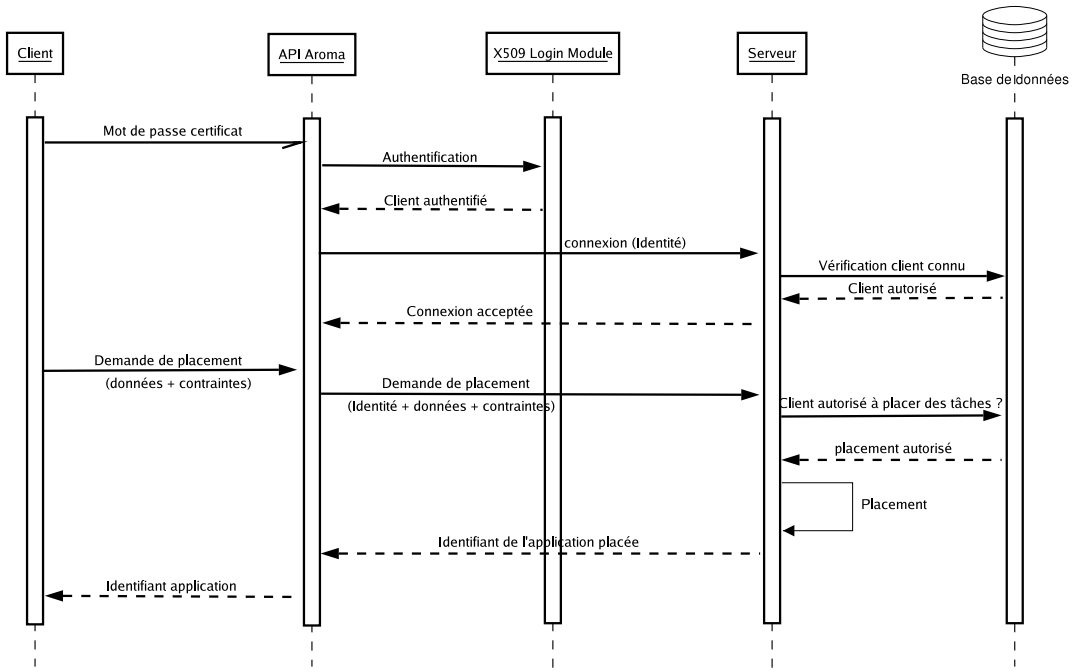


Fig. 5.5 – Vérification des droits lors d'une connexion via l'API

seuls les modules fournissant les fonctionnalités pour lesquelles l'utilisateur possède une autorisation, sont visibles. Ces modules sont appelés *plugins* graphiques.

5.4.3.1 Définition des *plugins* graphiques

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

Dans la version actuelle d'Aroma, il existe sept *plugins* différents :

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

ou de visualiser l'état courant d'une grille ou d'un sous-ensemble de machines d'une grille.

Chacun de ces services est accessible à partir de l'interface graphique cliente, sauf le *Services Launcher* qui correspond à l'interface graphique elle-même. Chaque *plugin* est ensuite lancé dans une nouvelle fenêtre (figure 5.6).

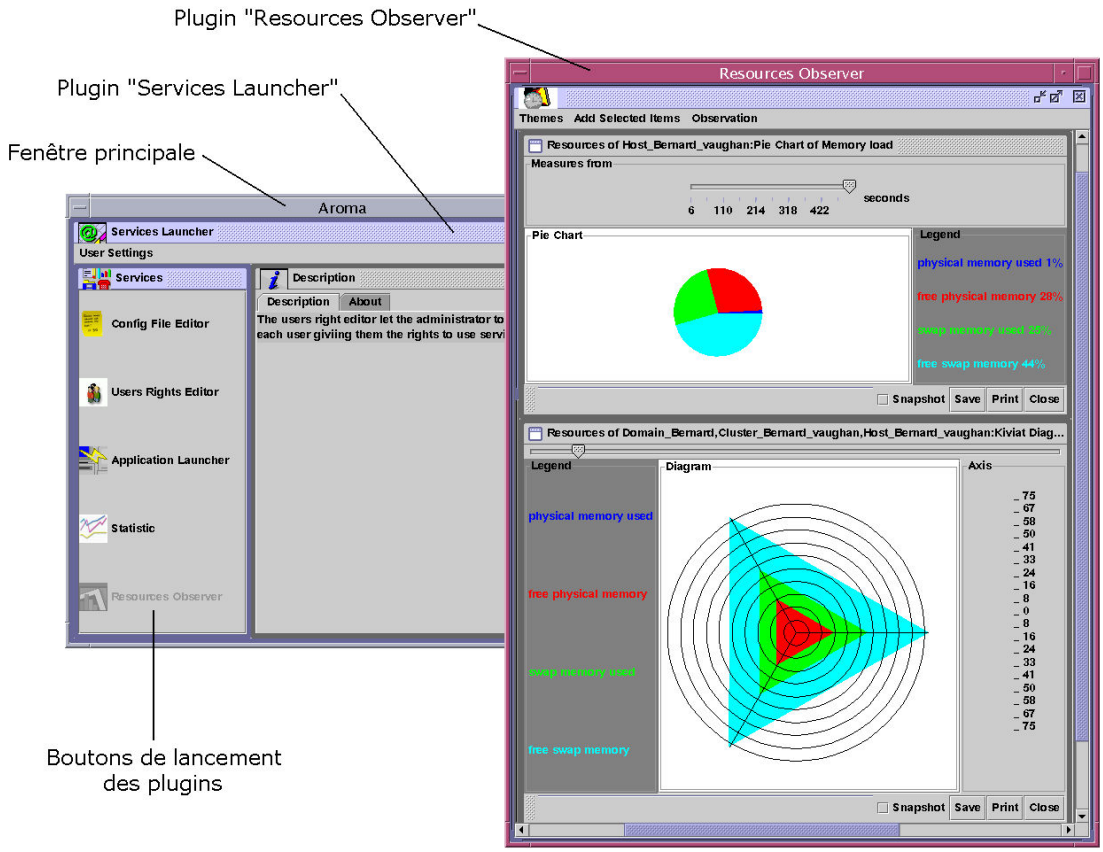


Fig. 5.6 – Interface graphique cliente

5.4.3.2 *Plugins* et droits utilisateurs

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

Le principe de fonctionnement est très simple (figure 5.7). Une table de la base de données d'Aroma contient, pour chaque utilisateur, la liste des *plugins* graphiques qu'il est autorisé à utiliser. Lorsque l'interface se connecte à un serveur (1), celui-ci accède à la base de données (2) et récupère la liste des *plugins* graphiques pour l'utilisateur connecté (3). Cette liste est transmise au client (4) qui vérifie s'il possède bien toutes les archives JAR nécessaires, et si les versions des *plugins* qu'il possède ne sont pas obsolètes. Dans le cas où un *plugin* graphique

est absent ou trop ancien, ce dernier est téléchargé depuis le serveur sur le poste client (5 et 6).

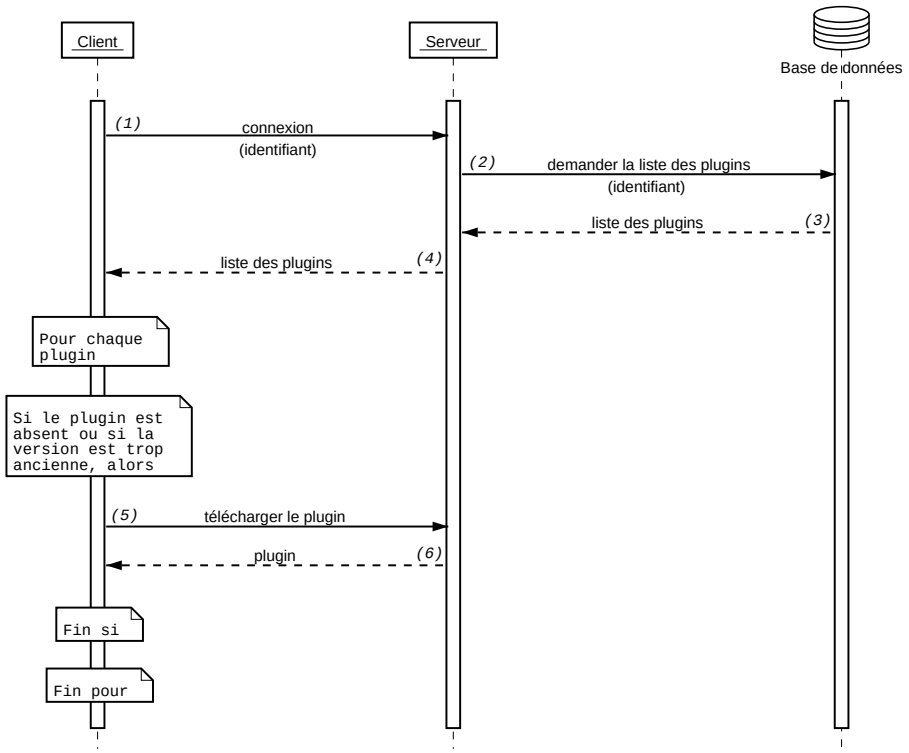


Fig. 5.7 – Principe de fonctionnement du téléchargement de *plugins* graphiques

On peut ainsi constater que, côté client, le chargement de *plugins* se fait à chaud au lancement de l'interface. Il n'y a pas besoin de modifier le code du client pour effectuer une mise à jour des *plugins*.

5.4.4 Chargement dynamique des plugins

Le chargement dynamique de *plugins* graphiques est très similaire au chargement à chaud de capteurs de charge (section 4.6). Le principe général de fonctionnement est le même dans les deux cas.

Ainsi, des fichiers de configuration écrits en XML, dans lesquels les *plugins* graphiques disponibles sont décrits, sont utilisés :

- un fichier par *plugin* graphique contenant diverses informations, telles que :
  - le nom du *plugin* graphique,
  - le nom de son développeur,
  - son numéro de version,
  - le nom de la classe qui implémente la fenêtre du *plugin* graphique,
  - une description du *plugin* graphique,
  - l'emplacement de l'icône à afficher sur le bouton de lancement du *plugin* graphique.
- un fichier contenant la liste des *plugins* graphiques disponibles. Il contiendra :
  - le nom du *plugin* graphique,

– le nom de l'archive JAR qui le contient,
– le nom du fichier XML qui lui est associé.

Les informations contenues dans ces fichiers permettent de créer dynamiquement les instances de la classe `Description`, qui représente une description graphique du service (icône, résumé du service rendu, numéro de version...).

Deux classes sont mises en place à cette occasion : la classe `PluginsListReader` qui récupère la liste des *plugins* disponibles, et la classe `PluginDescriptionReader` qui récupère les informations sur chacun d'entre eux. Ces deux classes s'interfacent avec les classes générées par JAXB pour lire les fichiers XML. Ainsi, les *plugins* peuvent être chargés à la demande de façon dynamique, d'après le contenu des fichiers de configuration.

Le diagramme de la figure 5.8 montre le fonctionnement du système mis en place. Tout comme le chargement des capteurs de charge, le chargement des *plugins* est initié par le serveur. Ainsi, le déclenchement de cette opération est effectué dans la classe `AromaServiceImpl`. Ensuite, les fichiers de configuration sont lus afin de pouvoir charger les nouveaux plugins (création d'une nouvelle instance de la classe `Description`). Enfin, la classe `Authority` reçoit la nouvelle liste des *plugins* disponibles.

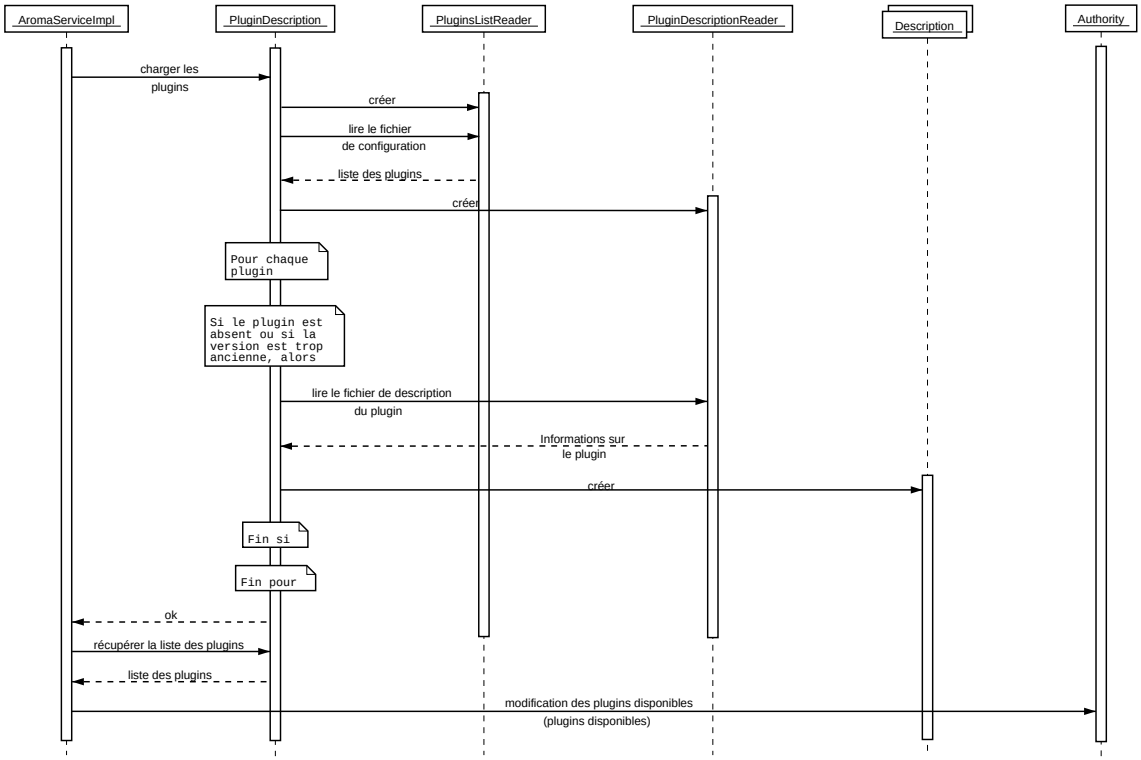


Fig. 5.8 – Chargement des *plugins* graphiques au sein des serveurs

5.5 Portage d'une application existante en mode ASP.

Les différentes modifications à apporter à une application en vue de son utilisation en mode ASP vont maintenant être décrites.

Nous utiliserons, pour illustrer nos propos, un logiciel de simulation et d'optimisation de réseaux de télécommunications, dont le portage a été réalisé dans le cadre du projet CASP[60].

5.5.1 Concept général

Le logiciel porté étant développé en langage C++, il a été choisi de développer un composant intermédiaire appelé *interacteur* chargé de faire le lien entre le logiciel et le gestionnaire de ressources Aroma (figure 5.9). Ce composant dialogue avec le logiciel existant en utilisant la Java Native Interface (JNI).

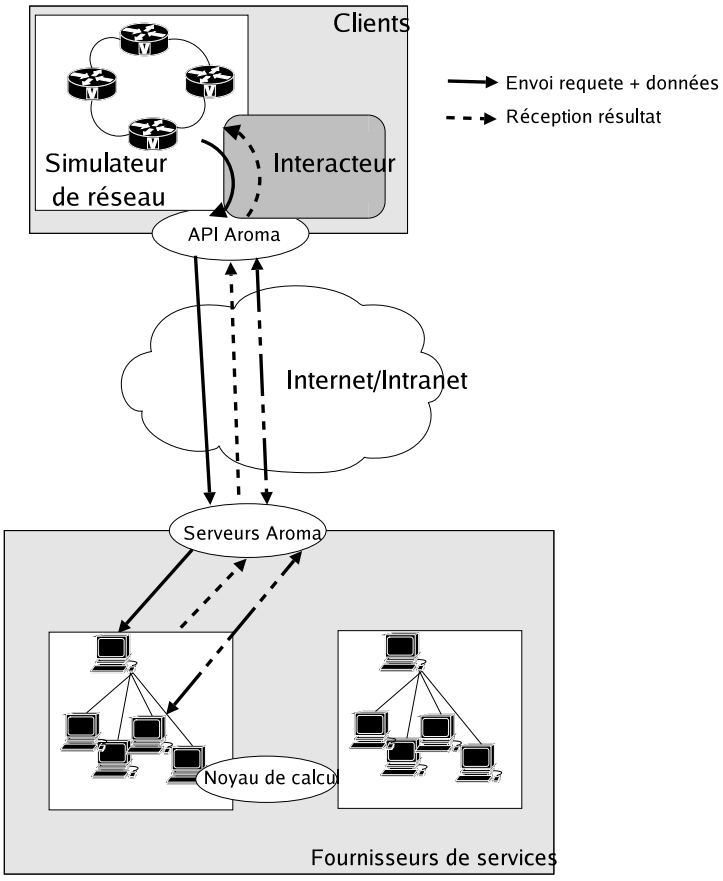


Fig. 5.9 – Rôle de l'interacteur

Les fonctionnalités que fournit l'interacteur sont :
- la soumission de travaux,
- la gestion des utilisateurs du contrat.

5.5.2 Authentification du client.

La première opération à réaliser, pour utiliser le logiciel en mode ASP, est l'authentification du client et la connexion au gestionnaire de ressources. Cette étape peut, soit remplacer l'étape d'authentification du logiciel pré-existant (dans le cas où une authentification était nécessaire), soit être effectuée juste avant la soumission de la requête distante. C'est cette

dernière solution qui a été retenue, puisqu'aucune identification n'était nécessaire pour utiliser le logiciel existant.

Ainsi, l'utilisateur continue d'utiliser le logiciel de manière standard : dessin d'une topologie réseau, calcul des matrices de routages... Puis, lorsqu'il souhaite réaliser un calcul lourd, tel que l'évaluation de la qualité de service (GoS) du réseau par exemple, l'opération d'authentification et de connexion au gestionnaire de ressources est réalisée. Cette opération est effectuée en faisant appel à la méthode *connect()* de l'API d'Aroma. Cette méthode affiche un *Callback Handler* permettant de saisir :

- le login de l'utilisateur,
- le mot de passe de l'utilisateur,
- le contrat auquel l'utilisateur appartient,
- l'identifiant de la grappe que le client souhaite utiliser.

La phase d'authentification utilise le certificat numérique de l'utilisateur dont l'emplacement doit être indiqué dans un fichier de configuration qui doit être accessible depuis la machine du client. Une fois l'authentification réussie, une étape de connexion avec le gestionnaire de ressources est réalisée afin de vérifier que l'utilisateur est bien autorisé à se connecter à la grappe de machines indiquée.

5.5.3 Calcul distant

Une fois la phase de connexion passée avec succès, l'utilisateur peut soumettre sa tâche de calcul. Pour cela, il doit indiquer le nombre de processeurs qu'il souhaite utiliser, sélectionner les fichiers à utiliser pour le calcul, et indiquer une prédiction du temps processeur consommé par chaque tâche de calcul. L'ensemble de ces informations est passé en argument de la méthode *schedule()* de l'API Aroma. Des contraintes supplémentaires telles qu'une date butoir ou des limitations sur la taille mémoire des machines devant être utilisées peuvent être également passées en argument de cette méthode.

La méthode *schedule* renvoie une clé unique permettant par la suite, de suivre l'état d'avancement de la requête, de stopper l'exécution de la requête ou de récupérer les résultats de l'exécution de la requête. Les fichiers résultats sont téléchargés à l'emplacement indiqué par le client lors de son appel à la méthode *schedule*.

5.5.4 Gestion du contrat

Un certain nombre de méthodes de l'API permettent de visualiser et de gérer le contrat de l'utilisateur. Les opérations réalisables diffèrent selon que l'utilisateur est administrateur du contrat ou simple utilisateur. Un simple utilisateur peut réaliser les actions suivantes :

- afficher les termes du contrat,
- afficher les droits associés au contrat : limites et valeur consommée pour chaque ressource de la grappe,
- afficher ses propres droits,
- modifier son mot de passe.

Un administrateur de contrat peut, en sus des opérations accessibles par un simple utilisateur :

- visualiser l'ensemble des utilisateurs de son contrat,
- créer de nouveaux utilisateurs du contrat,

– modifier les droits d'un utilisateur du contrat (dans la limite des droits du contrat).

5.6 Conclusion

Les principales difficultés liées à l'exploitation industrielle d'un gestionnaire de ressources en mode ASP ont été présentées. Aroma, dans le cadre du projet CASP et de la mise en ASP d'un logiciel de simulation et d'optimisation de réseaux de télécommunications, apporte des éléments de réponse aux problèmes posés.

Les mécanismes de sécurité mis en place permettent d'assurer la confidentialité des échanges et visent à préserver l'intégrité du code et des données. Cependant, ces mécanismes restent dépendants de la sécurité locale de chaque machine, et nécessitent de faire confiance aux administrateurs locaux des machines d'exécution. Ces limitations ne sont pas problématiques dans le cas d'utilisation de grilles de taille réduite, gérées par un seul fournisseur de services, comme cela était le cas dans le projet CASP. Cependant, dans le cas de grilles de taille plus importante faisant appel à plusieurs fournisseurs de services, des solutions de chiffrement des données écrites sur le disque, de signature numérique du code et d'utilisation de *sandbox* lors de l'exécution du code doivent être envisagées [49].

La notion de contrat, permettant de limiter l'utilisation des ressources, et plus généralement la notion de modèle économique autour des grilles de calcul est également une problématique intéressante. Le projet RNTL CASP a permis, grâce à la collaboration d'acteurs industriels, de faire une première étude de ce problème. Aroma offre différentes fonctionnalités permettant de modeler le comportement du gestionnaire de ressources, en fonction de l'utilisateur, et de développer différents modèles économique autour des grilles[5].

Chapitre 6

Évaluation de performances d'applications distribuées.

Sommaire

6.1 Introduction

Le but d'un environnement ASP est d'offrir un accès cohérent et performant à un ensemble de services. Un même portail ASP peut permettre à ses utilisateurs d'exécuter des simulations déportées sur les machines du fournisseur de services, d'accéder à des contenus multimédias mis à disposition par le fournisseur, de gérer son compte ASP... Ainsi, une même grappe de machines peut servir, par exemple, à la fois de support d'exécution à des applications de calcul, héberger des contenus multimédias et servir de serveur HTTP.

Une gestion des ressources, même optimale, ne peut garantir à elle seule une qualité de service satisfaisante. Pour satisfaire ce besoin, une étude préalable doit être menée afin de dimensionner les grappes de machines à utiliser.

Dimensionner une grappe de machines, signifie déterminer le nombre et la puissance des machines constituant cette grappe, ainsi que le débit et la topologie des réseaux reliant ces machines, afin d'offrir une capacité de traitement permettant de fournir des résultats aux clients dans des délais acceptables. Une telle problématique est une opération complexe qui nécessite la caractérisation du comportement des clients de chaque type de service, l'étude du comportement de chaque famille d'applications, la modélisation du fonctionnement des machines et des réseaux d'inter-connexion.

De nombreux travaux concernant la modélisation de réseaux de télécommunication ont été menés au LAAS durant ces vingt dernières années. Ces travaux ont abouti à la création d'un simulateur hybride distribué DHS (Distributed Hybride Simulator) permettant la conception et le dimensionnement de réseaux IP. Un des objectifs de cette thèse est d'enrichir ce simulateur afin de l'utiliser pour le dimensionnement de grappes de calcul. Le travail présenté ici concerne spécifiquement la modélisation du comportement des utilisateurs, du comportement des applications et du fonctionnement des machines, et l'intégration de ces différents modèles au sein du simulateur de réseau pré-existant.

Dans un premier temps, un panorama des différentes techniques utilisées pour modéliser l'exécution d'applications ainsi que les concepts de base du simulateur DHS seront présentées. Les différents modèles mis au point dans le cadre de cette thèse et leur intégration au sein du simulateur DHS seront ensuite détaillés.

6.2 L'évaluation de performance d'applications.

Trois principales techniques sont couramment utilisées lorsqu'il s'agit d'évaluer les performances de systèmes complexes : les benchmarks, la simulation, et les méthodes analytiques. Ces trois techniques diffèrent par la qualité de la prédiction obtenue, le délai d'obtention de cette prédiction et la simplicité d'utilisation et de conception du modèle.

6.2.1 Les benchmarks

6.2.1.1 Présentation

Les techniques de benchmarks reposent sur des analyses métrologiques d'applications réelles ou d'applications spécifiquement créées pour l'évaluation de performances. Elles consistent à exécuter une application de référence sur une machine réelle et à mesurer les performances du couple application/machine.

Ces techniques s'avèrent le plus souvent très précises puisqu'elles reposent sur une exécution réelle de l'application, mais sont également délicates à mettre en œuvre. En effet, elles nécessitent de disposer à la fois des machines support d'exécution, et d'instrumenter l'application afin d'obtenir des indices de performance pertinents. Le délai d'obtention des résultats peut également représenter un frein à l'utilisation de ces techniques. Ce délai correspond en effet, au temps réel d'exécution de l'application, qui est le plus souvent non négligeable. De plus, il est toujours délicat de certifier que l'instrumentation du système étudié ne modifie pas le comportement temporel de ce système, falsifiant ainsi les résultats obtenus.

6.2.1.2 Les outils existants

Le *Standard Performance Evaluation Corporation (SPEC)*[68] maintient de nombreux outils de benchmarks standardisés, ayant pour but de représenter le comportement de différents types d'applications. Ces outils sont régulièrement mis à jour afin de représenter, de la manière la plus fiable possible, le comportement des applications actuelles. Ces techniques sont couramment utilisées pour dimensionner un serveur devant répondre aux requêtes d'une unique application (type serveur Web, par exemple) ou pour comparer les performances de différentes machines (machines parallèles, par exemple). Cependant, dans le cadre de grappes de machines où un grand nombre de machines hétérogènes servent de support d'exécution à plusieurs types d'applications ayant des caractéristiques différentes, une telle approche n'est pas envisageable car trop complexe à mettre en œuvre et trop lente.

6.2.2 La simulation

6.2.2.1 Présentation

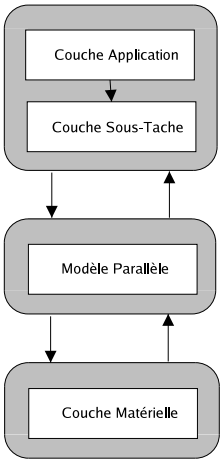
Simuler un système signifie construire un modèle du comportement de ce système, et étudier le comportement de ce modèle lorsqu'il est soumis à une certaine charge de travail. La simulation possède l'avantage de pouvoir s'appliquer à tous types de problèmes. Cependant certaines précautions doivent être prises afin de garantir l'efficacité de cette technique :

- Il est nécessaire de définir les parties du système à simuler et avec quel niveau de détails. Il est en effet inutile de perdre du temps à détailler le comportement de certaines parties du système si ces dernières n'ont qu'une faible influence sur le résultat qui nous intéresse. Un trop grand niveau de détails augmente également le risque d'erreurs.
- Un grand nombre d'informations peuvent être fournies par une simulation. Il est nécessaire de cibler les informations que l'on souhaite obtenir afin de les extraire, par des méthodes statistiques par exemple, de la masse d'information fournie par la simulation.
- Le modèle doit pouvoir être implémenté le plus efficacement possible afin de garantir un temps de simulation le plus court possible.

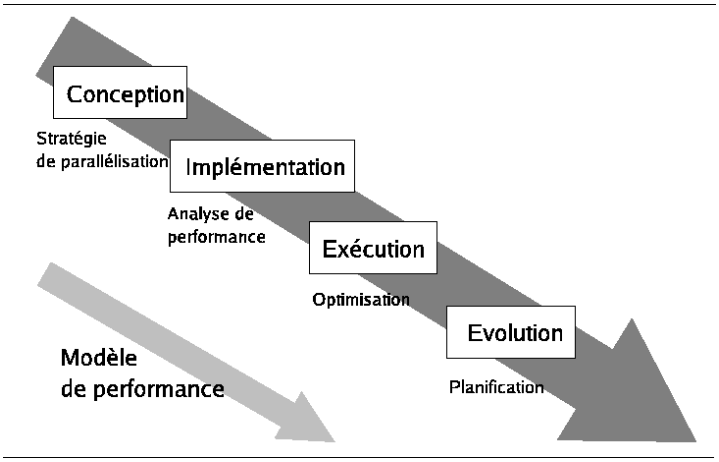
6.2.2.2 Le simulateur PACE

L'outil PACE (Performance Analysis and Characterization Environment)[69][55] développé à l'université de Warwick est un outil de simulation permettant de prédire les performances de systèmes parallèles et distribués. La méthodologie utilisée repose sur l'utilisation de trois modèles organisés en couches (figure 6.1(a)). Un modèle de l'application permet de caractériser l'application simulée : nombre de tâches, paramètres de l'application permettant de modifier la taille du problème... Un modèle concernant l'aspect parallèle de l'application

décrit les relations de précédence ainsi que les communications entre les différentes tâches de l'application parallèle. Enfin, un modèle concernant l'aspect matériel permet d'exprimer les caractéristiques des ressources physiques de la machine : processeur, bande passante réseau, charge, hiérarchie des caches mémoire ...



(a) Les trois modèles de PACE



(b) Rôle de l'évaluation de performances au cours du cycle de vie du logiciel

Fig. 6.1 – L'outil PACE

Chacun de ces trois modèles peut être décrit en utilisant différents niveaux d'abstraction. Par exemple, le traitement d'une fonction écrite en C pourra être représentée soit par une durée temporelle, soit par un nombre d'opérations flottantes. Ces différents niveaux d'abstraction permettent d'utiliser l'environnement PACE à différentes étapes du cycle de vie d'un logiciel (figure 6.1(b)).

L'environnement PACE vise deux principaux types d'utilisations : l'utilisation hors ligne et l'utilisation en ligne. L'utilisation hors ligne est destinée à prédire ou à étudier le comportement d'une application. Des exemples d'utilisation sont la prédiction du temps d'exécution lorsque la taille du problème ou les caractéristiques du support d'exécution évoluent, le choix de la stratégie de parallélisation d'une application ou l'étude du passage à l'échelle d'un couple application/machine. La durée d'obtention de la prédiction n'est pas, dans ces cas d'utilisation, un critère prépondérant ce qui autorise l'utilisation de modèles très détaillés. Le mode d'utilisation en ligne désigne l'utilisation de l'outil d'évaluation de performances au cours de l'exécution réelle de l'application, afin d'optimiser son temps de traitement. Les deux principaux exemples d'utilisation de ce mode sont :

- l'optimisation du comportement d'une application (choix des ressources les plus performantes, réutilisation des caches ...),
- l'optimisation du comportement d'un ensemble d'applications (ordonnancement de tâches, maximisation de l'utilisation des ressources ...).

L'utilisation en ligne impose un temps de simulation très bref (traitement interactif), par contre la précision de la prédiction n'est pas un critère primordial.

6.2.2.3 OPNET

Le simulateur OPNET[56] propose un ensemble d'outils permettant d'étudier et de prédire les performances de bout en bout d'applications.

L'outil ACE (End-to-end Application Analysist) permet d'étudier des traces d'exécutions d'applications distribuées. Cet outil permet de visualiser les communications entre les différentes entités d'une application distribuée avec différents niveaux d'abstraction : niveau messages échangés entre les entités jusqu'au niveau paquet. Un module de diagnostique permet ensuite de détecter de manière automatique les principaux goulots d'étranglements de l'application distribuée, à partir de l'étude des communications ayant le plus d'impact sur le temps total d'exécution de l'application. Enfin, un dernier module permet d'évaluer les gains substantiels engendrés par l'utilisation d'équipements plus performants ou par la modification du comportement de l'application.

L'outil SSM (Server Specialized Models) permet, quant à lui, d'étudier les performances des serveurs. Il se base sur une batterie de tests de performances des différents serveurs des principaux vendeurs du marché (Dell, IBM, Sun et HP), et de différents composants de références (différents modèles de disques durs, par exemple) afin de prédire les performances d'un serveur cible soumis à une charge de travail pré-définie.

6.2.3 Méthodes analytiques

6.2.3.1 Présentation

Les méthodes analytiques possèdent l'avantage d'être rapides et d'offrir une vision simplifiée du comportement du système. Cependant l'ensemble des problèmes modélisables de manière totalement analytique est de taille réduite. De ce fait, représenter le comportement détaillé d'un système devient rapidement impossible.

6.2.3.2 Modèles stochastiques :

De nombreuses études à base de modèles stochastiques ont été menées dans les années 70 [42][12][41]. Cependant, ces études se limitent à l'étude de systèmes de taille réduite (une seule machine) ou à l'étude de phénomènes très précis (modélisation fine du comportement d'un disque dur, par exemple). Ces techniques sont difficilement utilisables lorsque l'on souhaite modéliser le comportement de l'ensemble d'une grappe de machines, impliquant la modélisation de plusieurs machines, du réseau et de plusieurs applications.

6.2.3.3 Méthodes "historiques" :

Plus récemment, certaines techniques appelées méthodes "historiques" [1][20][3] ont été développées dans le but d'aider à l'ordonnancement de tâches dans les grilles de calcul. Ces méthodes "historiques" reposent sur l'analyse détaillée de mesures de performances issues de traces réelles d'exécutions d'applications sur différentes machines et avec différents paramètres d'entrée. L'objectif est de discerner les éléments de l'application, ou de la machine, ayant un impact significatif sur les performances du système, afin de déterminer un ensemble d'équations permettant de prédire efficacement la durée d'exécution d'une application sur une machine donnée. Ces prédictions sont ensuite utilisées lors du placement d'une application sur une grille de calcul, afin d'évaluer le coût d'exécution de cette application sur chacune des machines de la grille.

6.2.3.4 Layered Queuing Networks :

La méthode des *Layered Queuing Networks* [64] est une extension des réseaux de files d'attentes dont le but est de permettre la modélisation du phénomène de possession simultanée de ressources.

Cette méthode se base sur une représentation des applications et des machines sous forme de tâches organisées en couches [17] (figure 6.2). Chaque tâche représente, soit un traitement particulier d'une application, soit une ressource physique (processeur, disque ...) et est modélisée par une file d'attente. Une tâche ne peut requérir le service que d'une autre tâche de niveau inférieur. Ce service peut être une requête synchrone, asynchrone ou à plusieurs phases (durant une première phase, la requête est synchrone, puis durant la seconde la tâche appelante est libérée alors que la tâche appelée doit réaliser un traitement supplémentaire).

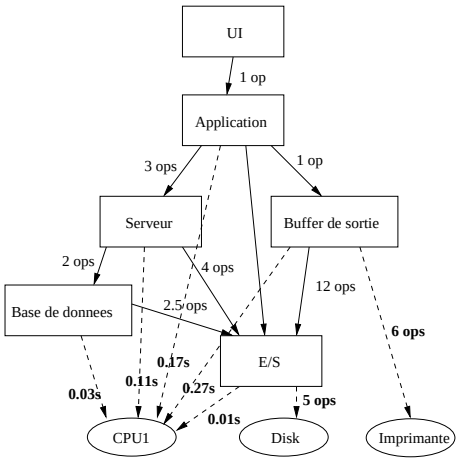


Fig. 6.2 – Exemple de LQN

Chaque tâche est modélisée par une file d'attente, ce qui permet d'exprimer la contention au niveau de chaque ressource applicative ou physique. Le réseau est traité comme une ressource physique, ayant un taux de service qui représente le délai de la transmission, et pour l'utilisation de laquelle les tâches applicatives entrent en compétition.

La "méthode des layers" est une extension des réseaux de files d'attente classiques qui permet de tenir compte du fait que le temps passé dans un serveur dépend du service d'un serveur d'une couche inférieure. De ce fait les appels synchrones peuvent être modélisés. Trois principaux type d'interactions sont modélisables par cette technique :

- les appels synchrones,
- les appels asynchrones,
- la délégation.

Cette méthode vise particulièrement l'évaluation de performances d'applications durant les phases de conception de logiciel. En effet, le mode de représentation en couche qu'elle utilise peut être aisément obtenue à partir du diagramme de classes d'UML. De ce fait, l'évaluation de performances peut être intégrée dès la phase de conception du logiciel.

6.3 Le simulateur DHS.

L'outil DHS [15] est un simulateur général de files d'attente et de réseaux à commutation de paquet de type IP intégrant les mécanismes de différenciation de services et la commutation MPLS. La particularité de ce simulateur réside dans la notion de simulation hybride, qui combine des modèles de simulation stochastique et de simulation analytique. Des modèles différentiels de files d'attente sont utilisés pour évaluer le comportement de chaque nœud analytique, et des simulations de Monte-Carlo sont utilisées pour évaluer certains nœuds dont le modèle différentiel n'est pas connu. Le concept de simulation hybride y est intégré rigoureusement, afin de combiner ces deux types de modélisation au sein d'une même simulation, et de proposer un équilibre entre précision des résultats et rapidité de simulation.

La structure générale du simulateur, et plus particulièrement les éléments intervenant lors de la simulation événementielle, vont maintenant être présentés.

6.3.1 La simulation événementielle

La simulation événementielle est une simulation de Monte-Carlo. Chaque élément du réseau est dupliqué un certain nombre de fois afin de réaliser des simulations indépendantes, appelées aussi trajectoires. L'évaluation quantitative des grandeurs (nombre de paquets présents dans le système, délais, probabilité de perte) est effectuée en calculant la moyenne (et la variance si nécessaire) sur l'ensemble des trajectoires. Pratiquement, cela implique que toutes les trajectoires sont menées en parallèle. Leur différence se situe uniquement lors de la génération des variables aléatoires. Les générateurs associés à chaque trajectoire ont des caractéristiques identiques, mais fournissent des réalisations différentes pour chacune d'elles afin d'obtenir une statistique correcte de l'ensemble.

Bien que tous les événements soient classés et stockés dans un unique échéancier global, la simulation événementielle peut être vue comme un ensemble d'algorithmes indépendants et décentralisés, ayant une cohérence globale grâce à leur inter-connexion réalisée par le routage.

6.3.2 Les événements

Chaque action menée par le simulateur est représentée par un événement particulier. La boucle principale du simulateur consiste à gérer un échéancier, constitué d'une liste d'événements classés par date d'action.

6.3.3 Le paquet

Le paquet est à la base de la simulation événementielle. Les paquets du simulateur DHS correspondent aux paquets générés par une interface réelle de niveau 2 (couche réseau).

Les paquets basiques :

Au minimum, les informations transportées par un paquet sont :

- le flux d'appartenance,
- sa classe de service,
- sa destination,
- la taille totale des données transportées (entête comprise),
- le numéro de la trajectoire à laquelle appartient le paquet,
- la date de création du paquet,

– le nœud et la file dans lesquels il se trouve.

Un paquet de ce type peut être créé par la plus simple des sources (un générateur simple), par exemple de distribution d'inter-arrivées issue d'un processus de poisson. Avant d'arriver à destination, ces paquets peuvent être perdus sur un nœud dont les files sont saturées ou parce qu'un algorithme particulier l'a décidé (RED, Token Bucket). La classe de service d'un paquet sert à décider la file dans laquelle le paquet doit être inséré (classification) en entrée de chaque nœud.

Les paquets spéciaux :

Les paquets peuvent également être issus d'une source spéciale, comme un protocole de transport (par exemple TCP), ou un protocole de routage (par exemple OSPF). Dans ce cas des informations spécifiques lui sont ajoutées pour le fonctionnement des algorithmes associés au protocole.

Par exemple le protocole OSPF, qui a été en partie intégré dans le simulateur, a besoin d'échanger des informations sur les *link-states* entre routeurs. Ces informations sont transportées dans ces paquets spéciaux. De même, les paquets issus du protocole TCP contiennent le même type d'information que celles qui sont contenues dans l'entête TCP (les champs seq, ack, rwnd...).

Les paquets spéciaux subissent les mêmes traitements que les paquets classiques, mais sont transférés à une fonction associée au protocole avant d'être détruits une fois qu'ils sont arrivés à destination. Le protocole OSPF pourra exploiter les données transportées, tandis que le protocole TCP pourra renvoyer d'autres paquets spéciaux, par exemple de type TCP-ack, vers la source. La fonction de traitement associée à chaque type de paquet spécial est intégrée dans la définition du paquet lui-même, ce qui laisse une grande latitude sur l'étude de nouveaux protocoles et de leurs effets sur les performances d'un réseau.

6.3.4 Les nœuds, les flux et les *flowstates*

Afin de permettre la simulation de tout type de réseau, le simulateur DHS manipule différents objets qui sont le nœud et les inter-connexions de nœuds, le flux, et les paramètres de routage.

– Le nœud : le modèle de nœud utilisé dans le simulateur est un modèle général permettant de représenter tout type de files d'attentes et tout type d'ordonnancement (scheduling). Un intérêt majeur de ce simulateur pour les réseaux IP multiservices est d'avoir un modèle de nœud se rapprochant le plus de l'interface utilisée dans ce type de réseaux. Les mécanismes de *shaping, policing,* et *buffer management* concernant ces réseaux sont également introduits dans le modèle.

– Le flux : les caractéristiques d'un flux sont le nœud origine, le nœud destination, les paramètres de la source et la classe de service à laquelle il appartient. Une source est représentée par un générateur de paquets. Elle est caractérisée par la distribution des temps inter-paquets et des tailles de paquets. Des générateurs aléatoires de distribution voulue sont utilisés pour produire des dates d'arrivées et des tailles de paquets pour la simulation événementielle.

– Le *flowstate* (l'état du flux) : pour pouvoir mesurer la charge induite par les flux dans une file d'attente, une représentation de l'état de chaque flux en chaque nœud est nécessaire. Cela permet de fournir la charge, le taux de perte, les délais et le débit de sortie de chaque flux en chaque nœud. Cette représentation est appelée *flowstate*.

Un *flowstate* est associé à un flux et un nœud traversé par ce flux. Ce dernier est calculé par la statistique des trajectoires événementielles des nœuds simulés.

6.3.5 Le routage

Le routage permet d'aiguiller les paquets entre les différents nœuds du réseau simulé.

6.3.5.1 Structure du routage

Le routage est modélisé par un poids $r_{i,j}^{f}$ affecté au flux f, dans le nœud i, sur le lien le connectant au nœud j. Pour un flux donné, la somme des poids de tous les liens de sortie vaut 1. Le partage de charge sur les chemins est ainsi possible. Ce type de représentation est suffisamment général pour permettre :

- Le routage statique par flux : il s'agit du routage à la fois le plus simple et le plus général. Il est calculé avant, et peut être mis à jour pendant la simulation. Il s'agit de positionner les coefficients $r_{i,j}^{f}$ pour que chaque flux f suive un chemin prédéterminé. Ce routage est le plus fin que l'on puisse avoir dans le simulateur (chaque flux peut être partagé sur plusieurs routes différentes).
- Le routage par destination (utilisé dans Internet) : en utilisant le routage statique par flux, il suffit d'affecter des coefficients égaux à 1 aux liens de sortie choisis, et 0 aux autres. Ainsi le routage IP peut être modélisé.

Le routage dynamique est possible, car les coefficients de partage peuvent être modifiés par un algorithme extérieur (pré-calculé ou non).

Sur un réseau avec beaucoup de trafic, le routage dynamique par flux peut engendrer une structure de données assez lourde. Une structure de données de type routage IP par classe de service est également possible. L'utilisation de ce type de routage a le même effet sur le simulateur que dans le réseau Internet : les tailles des tables de routage en chaque nœud diminuent considérablement par rapport aux tailles des structures de données nécessaires pour le routage par flux.

6.3.6 Les sources de trafic TCP

Le protocole TCP est simulé à l'aide de sources de trafic TCP. Une source de trafic doit être définie entre chaque interface du nœud origine et chaque interface du nœud destination d'un trafic TCP. La source de trafic regroupe deux flux, un flux pour les paquets allant de l'origine vers la destination, et un flux pour les paquets allant de la destination vers l'origine (figure 6.3). Lors de l'envoi d'un message entre l'origine et la destination, les paquets de données empruntent le flux origine vers destination, tandis que les paquets "d'accusés de réception" empruntent le flux opposé. La source de trafic génère les différents paquets (données et contrôle) en respectant l'algorithme du protocole TCP Version NewReno[35] (annexe B).

Le simulateur fournit des informations sur les délais de bout en bout et sur la gigue pour chaque flux.

6.4 Système modélisé

Notre objectif est de simuler le fonctionnement de grappes de machines utilisées en mode ASP. Le système modélisé (figure 6.4) est composé de plusieurs serveurs reliés entre eux

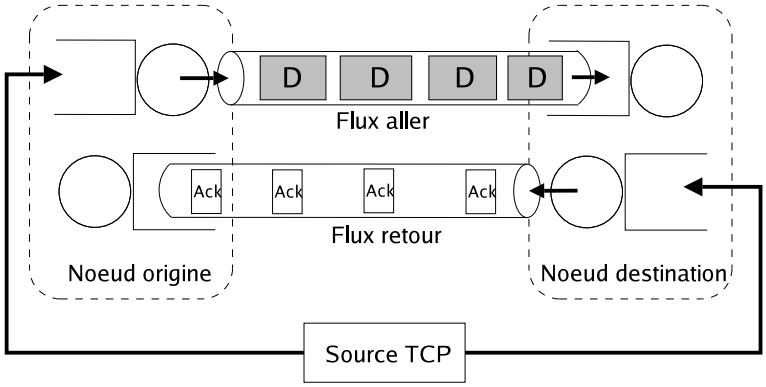


Fig. 6.3 – Source de trafic TCP

par des réseaux d'inter-connexion. Sur chacun de ces serveurs, plusieurs instances d'applications, ayant des caractéristiques propres et appartenant à des clients potentiellement distincts, peuvent s'exécuter simultanément. Un nœud particulier de la grappe joue le rôle de frontal du système ASP. Il répartit les applications sur les différents serveurs de la grappe, en utilisant une politique d'ordonnancement donnée, afin d'optimiser la qualité de service globale du système ASP.

Notre but est de pouvoir dimensionner les grappes de machines du fournisseur d'accès. L'objectif est de pouvoir visualiser le comportement du système face à un changement de trafic ou de prédire l'impact d'une modification d'un des composants des grappes de machines sur les performances globales du système.

Différents types d'applications sont concernés par ce type d'environnements : des applications de calcul (séquentielles ou parallèles) et des serveurs applicatifs donnant accès à des contenus multimédias interactifs : serveurs webs de pages statiques et dynamiques, et serveurs d'applications multi-tiers. Les travaux présentés dans cette thèse se limitent à la modélisation d'applications de calcul et de serveurs Web présentant des contenus statiques.

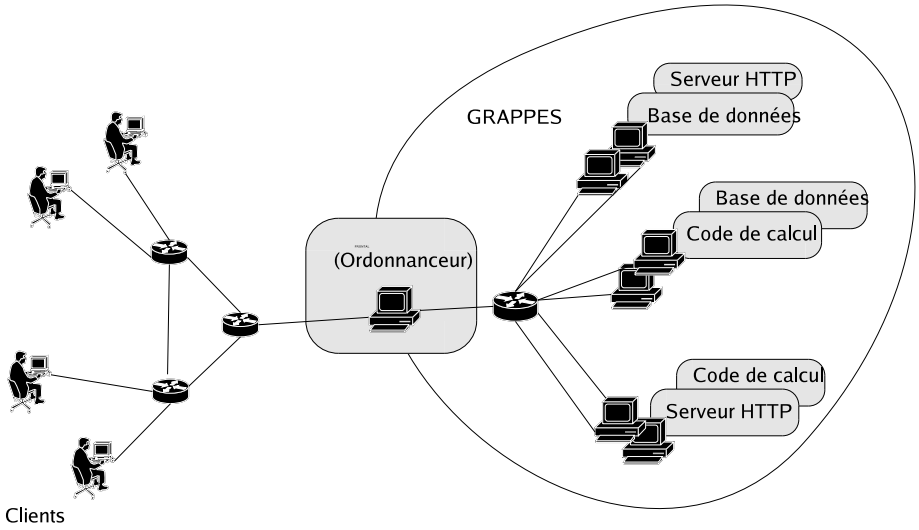


Fig. 6.4 – Système modélisé

Nous allons maintenant analyser le comportement des différents acteurs ayant une influence sur les performances du système étudié, et présenter les différents modèles mis en place dans le simulateur.

6.4.1 Comportement des clients

Le comportement d'un client diffère selon qu'il accède à une application de calcul ou à une application interactive.

6.4.1.1 Client d'une application de calcul

Un client du fournisseur de service réalise, au cours de la durée de vie de son contrat, un certain nombre de requêtes vers une ou plusieurs applications de calcul. Les différentes requêtes d'un même client, qu'elles concernent la même application ou des applications différentes, sont considérées comme étant indépendantes. En effet, un client peut très bien soumettre un nouveau calcul, sans être obligé d'attendre la terminaison de sa requête précédente (figure 6.5). De ce fait, du point de vu de notre modèle, un client correspond à une instance d'application, c'est à dire l'exécution d'une application donnée sur une ou plusieurs machines de la grappe.

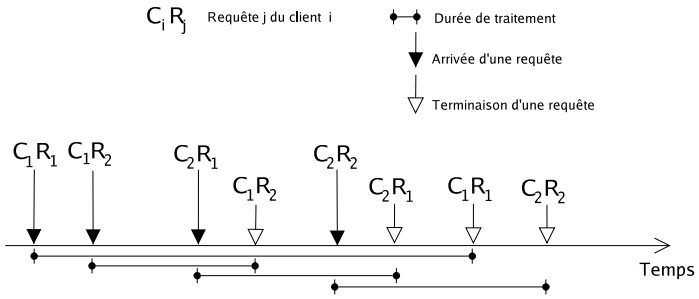


Fig. 6.5 – Comportement des clients d'applications de calcul au cours du temps

Ainsi, la charge de travail demandée à la grappe de machines par des requêtes du type applications de calcul est modélisée par le nombre de clients, le nombre d'exécutions par client et le temps d'inter-arrivée entre deux requêtes de calcul du même client. Chaque requête de calcul représente une charge de travail déterminée en fonction de l'application concernée et des paramètres de cette application, représentés par le modèle application.

6.4.1.2 Client d'une application interactive

Le comportement d'un client d'une application interactive est caractérisé par une session, elle-même définie par une suite de requêtes ayant un ensemble de paramètres et une suite de période de réflexion (pauses) entre les réponses à ces requêtes (figure 6.6).

Le profil d'une session est donné par les distributions liées au nombre de requêtes par client et au temps de réflexion entre chaque requête. Le comportement des clients d'applications interactives vis à vis du système ASP est caractérisé par le nombre de clients et par le temps d'inter-arrivée entre les sessions d'un même client.

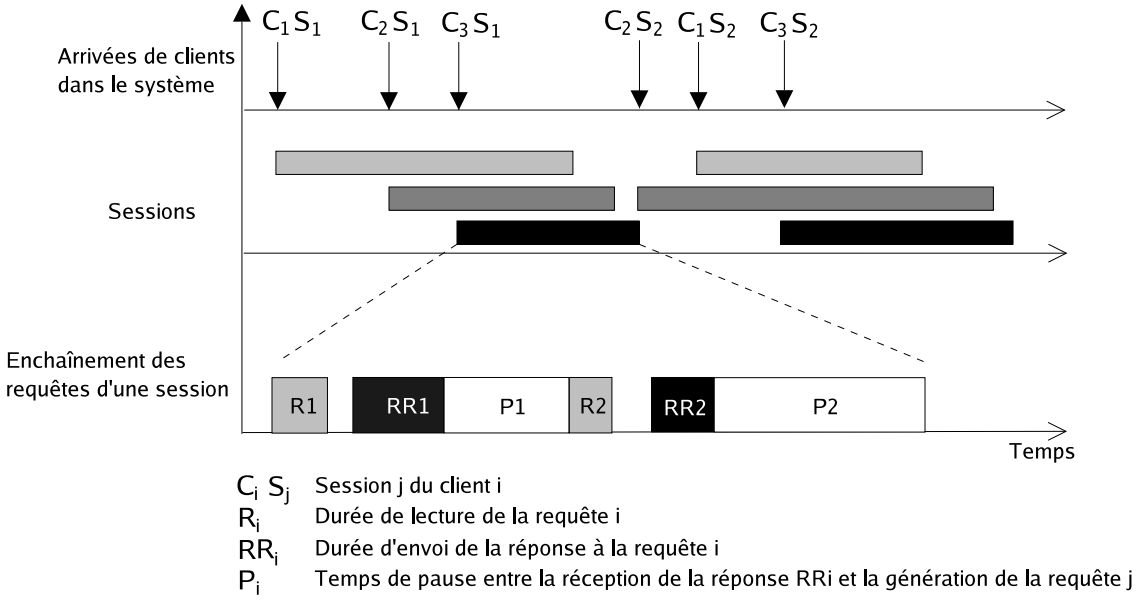


Fig. 6.6 – Comportement des clients d'applications interactives au cours du temps

6.4.2 Caractéristiques des applications

Un portail ASP peut regrouper à la fois des applications conçues par le fournisseur de services (des codes de calcul, par exemple) pour lesquelles le fonctionnement interne est connu avec beaucoup de détails, des applications grand public (serveurs HTTP, serveurs d'applications) pour lesquelles seuls les principes de base du fonctionnement sont connus et des applications pour lesquelles la connaissance du fonctionnement est très limitée (codes de calcul appartenant à une autre organisation, par exemple).

Du fait de ces différents niveaux de connaissance des applications rencontrées, nous avons choisi de modéliser le comportement de l'application sous forme de flux de données. Les modèles d'applications ainsi obtenus seront renseignés par des données issues d'analyses métrologiques d'exécution antérieures de ces mêmes applications.

6.4.2.1 Application séquentielle

Le terme application séquentielle regroupe toute application dont les seules communications réseau se limitent aux interactions avec le client du système ASP (routine de calcul séquentielle, serveur HTTP simple ...). Une telle application peut être vue comme un enchaînement de phases d'accès aux différents composants d'un serveur : processeur, disque, mémoire, carte réseau... Le profil de l'application est alors donné par la taille de chacune de ces phases et par leur enchaînement. La figure 6.7 schématise la trace d'une application séquentielle i.

L'application i est caractérisée par :
- la distribution associée à la taille de la requête : R_i,
- la distribution associée à la taille du résultat renvoyé par le serveur : RR_i,
- la distribution du temps processeur demandé par une requête de type R_i : C_i,
- la distribution de la taille disque demandée par une requête de type R_i : D_i,

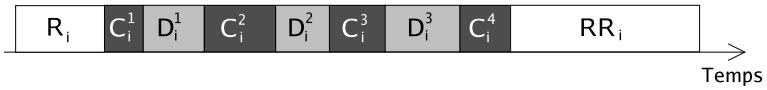


Fig. 6.7 – Trace d'exécution d'une application

– la distribution de la taille de mémoire vive consommée par une requête de type R_i : M_i.

Chacune des distribution caractérisant l'application peuvent être paramétrées, soit par la taille de la requête R_i, soit en fonction d'autres paramètres de l'application (paramètres fournis par l'utilisateur, par exemple la taille d'une matrice ...).

Le déroulement d'une application débute toujours par la lecture de la requête R_i, puis un enchaînement de phases de calcul et d'accès au disque, et enfin la phase d'envoi du résultat sur le réseau.

6.4.2.2 Serveur HTTP

Le téléchargement d'une page Web sur un serveur HTTP peut s'apparenter à l'exécution d'une application séquentielle. Cependant, un certain nombre de caractéristiques sur le comportement de ces applications sont connues et permettent de simuler plus précisément leur fonctionnement.

Le rôle du serveur HTTP est d'analyser les requêtes de chaque client et de renvoyer à chacun le ou les fichiers demandés. Un serveur HTTP doit traiter des requêtes venant de plusieurs clients simultanément. Pour ce faire, il doit créer plusieurs processus, ou plusieurs *threads* s'exécutant en parallèle.

Un certain nombre de paramètres limitent le nombre et le comportement de ces processus afin d'éviter la saturation de la machine d'une part, et de limiter l'effet d'éventuelles fuites mémoires d'autre part. Les principaux paramètres du serveur Apache, pris en compte par le simulateur sont :

- `MaxClients` : limite le nombre de requêtes simultanées pouvant être traitées sur une machine.
- `MaxRequestsPerChild` : limite le nombre de requêtes pouvant être traitées par un même processus.
- `KeepAlive` : prise en compte des requêtes persistantes définies par la version 1.1 du protocole HTTP (voir section 6.4.3.1).
- `KeepAliveTimeout` : délai d'inactivité à partir duquel une connexion persistante sera fermée par le serveur.

6.4.2.3 Application parallèle

Deux types d'applications parallèles sont considérées dans ce manuscrit : les applications représentables sous forme de graphe et les applications maître/esclaves.

Applications représentables sous forme de graphe :

Ces applications sont modélisées par un graphe acyclique exprimant les dépendances entre

les différentes tâches (figure 6.8).

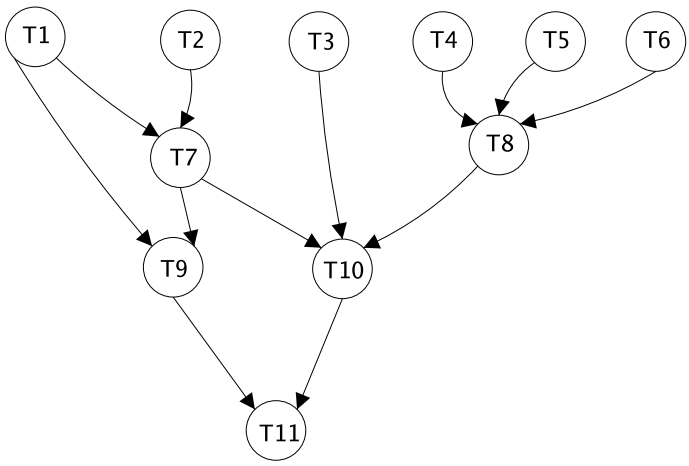


Fig. 6.8 – Application parallèle représentable sous forme de graphe

Chaque nœud du graphe représente une tâche de l'application parallèle. Le comportement d'une de ces tâche est similaire au comportement d'une application séquentielle : réception d'un message, enchaînement de phases de calcul et d'accès au disque, puis envoi d'un message.

Lors de la création d'une application parallèle, toutes les tâches de l'application sont créées sur les machines choisies par l'ordonnanceur du système ASP. Seules les tâches situées au sommet du graphe débutent leur exécution, les autres sont bloquées en attente d'un message. Chaque tâche fille du graphe ne débute son exécution que lorsqu'elle a reçu un message de la part de chacune de ses tâches parentes.

L'application parallèle est terminée lorsque toutes ses tâches ont terminé leur exécution. La tâche n'ayant aucun fils envoie le résultat du calcul au client du système ASP.

Applications maître/esclaves :

Une application maître/esclaves est caractérisée par une tâche particulière appelée maître et N tâches esclaves identiques (figure 6.9).
Le travail effectué par chacun des acteurs est le suivant :

Maître :

Tant que il y a des valeurs à calculer **Faire**
- envoi de points à calculer aux esclaves en attente de travail
- récupération des valeurs calculées par les esclaves
- calcul de résultats intermédiaires

Fin Tant que

Esclave :

Boucle
- attente de travail
- calcul
- envoi de résultat

Fin Boucle

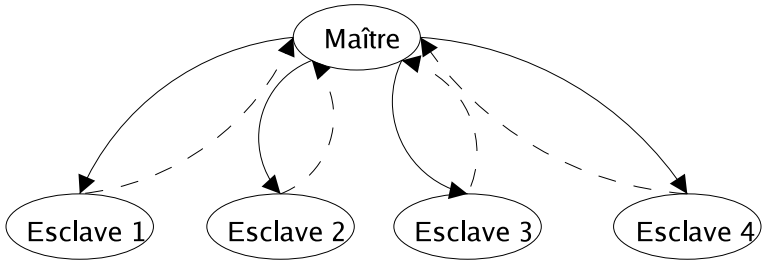


Fig. 6.9 – Application maître/esclaves

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

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

Une tâche esclave n'exécute son traitement que lorsqu'elle a reçu ses données de la tâche maître, et la tâche maître n'exécute son traitement que lorsqu'elle a reçu tous les messages des tâches esclaves.

Qu'il s'agisse d'applications maître/esclaves ou d'applications représentables sous forme de graphe, l'ordre de réception des messages par une tâche fille est un paramètre ayant une influence sur les performances globales de l'application. En effet, une tâche devant recevoir des messages de plusieurs autres tâches peut, soit traiter ces messages dans un ordre préétabli, soit les traiter dans leur ordre d'arrivée. L'ordre d'arrivée des messages dépend à la fois de l'état d'avancement des applications émettrices, et des performances du réseau. Cet ordre d'arrivée n'est pas prévisible et une politique de réception des messages par la tâche réceptrice dans un ordre fixe peut entraîner des ralentissements de la tâche réceptrice, de la tâche émettrice et des communications réseau (voir section 6.4.3.3). L'ordre de traitement des messages dépend à la fois de l'algorithme de traitement de la tâche réceptrice et des primitives de communication utilisées par l'application.

6.4.3 Les protocoles applicatifs

Les communications intervenants entre les différentes tâches d'une application parallèle, ou entre le client, le système ASP et les machines d'une grappe peuvent être de différentes nature. Il peut s'agir de simples messages asynchrones, de messages synchrones, dans le cas d'appels de méthodes distants par exemple, ou de protocoles de communications plus complexes. Les protocoles applicatifs doivent, de ce fait, être étudiés afin de faire le lien entre les besoins en communication des applications et les communications réseau.

6.4.3.1 Le protocole HTTP

Du point de vue logiciel, les trois acteurs principaux d'une communication Web sont le navigateur, le protocole HTTP et le serveur HTTP. Un client accède au Web en utilisant un navigateur, chargé d'échanger des informations avec les serveurs Web à l'aide du protocole HTTP (sous forme de requêtes et de réponses), et d'afficher les informations obtenues sous différentes formes (texte, images, vidéo ...). Le protocole HTTP, support des communications entre le client et le serveur repose dans la majorité des cas sur l'utilisation du protocole TCP/IP.

Deux versions du protocole HTTP sont couramment utilisées : les versions 1.0 [33] et 1.1 [34]. La principale différence, du point de vue de l'utilisation du réseau, entre les deux versions du protocole réside dans la manière dont les connexions TCP sont gérées.

- La version 1.0 du protocole utilise une nouvelle connexion TCP pour chaque objet transitant sur le réseau. Lorsqu'un utilisateur saisit une URL dans son navigateur, celui-ci établit une connexion TCP avec le serveur sur le port 80, puis il récupère le document souhaité en utilisant cette connexion. Une fois le document réceptionné par le client, la connexion TCP est fermée. Une nouvelle connexion sera créée pour chaque nouveau document demandé par le client. Ce mécanisme de création et destruction de connexion TCP à chaque téléchargement d'un nouvel objet est coûteux. D'une part, cela consomme du temps processeur sur le serveur, et d'autre part cela sous-utilise la bande passante réseau (phase de *slow-start* et ouverture bidirectionelle d'une connexion nécessitant 1 RTT à chaque connexion).
- La version 1.1 essaie de pallier les faiblesses de la version 1.0 en utilisant la notion de connexions persistantes. Toutes les requêtes provenant d'un même navigateur Web, à destination d'un même serveur Web utilisent la même connexion TCP. Cette connexion est fermée, soit par une requête spécifique du navigateur Web (fonctionnalité non implémentée dans la plupart des navigateurs Web), soit par le serveur Web lors de l'expiration d'un *timer* enclenché à chaque réception d'une requête.

6.4.3.2 Bibliothèques de passages de messages

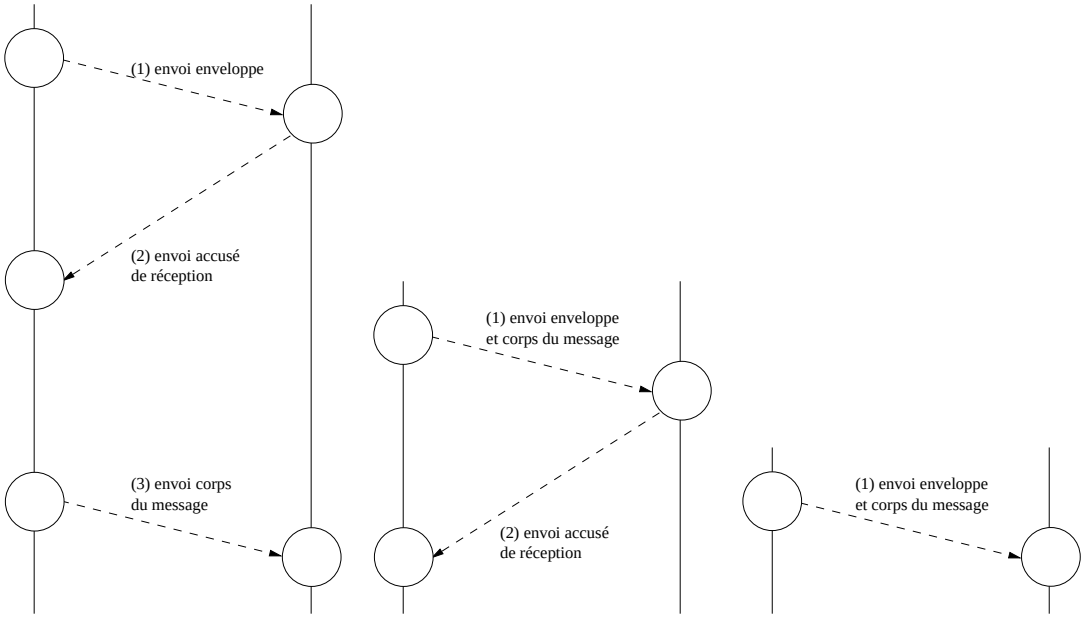
Les messages échangés par les applications parallèles utilisent des bibliothèques de passage de messages telles que MPI [13] ou PVM [16]. Ces bibliothèques définissent plusieurs modes de communication tels que les communications bloquantes, synchrones ou bufferisées. Ces communications peuvent utiliser plusieurs moyens de communication (différents protocoles de transport, mémoire partagée ...). Dans le cadre de notre étude, les messages échangés par les différentes tâches d'une application parallèle utilisent le protocole de transport TCP/IP[31][32].

La norme MPI définit deux principaux modes d'échanges de messages : les communications bloquantes et les communications non-bloquantes.

- Les communications bloquantes : lors d'un envoi bloquant, le processus initiant l'envoi est bloqué jusqu'à ce que la zone tampon dans laquelle le message est stocké soit libérée. Ce mode d'envoi permet de garantir la validité des données envoyées.
- Les communications non bloquantes : lors d'un envoi non bloquant, le processus initiant l'envoi est libéré immédiatement. L'utilisateur doit alors vérifier que les données ont bien été transmises avant de réécrire sur le même emplacement mémoire. Dans le cas

contraire, le message est écrasé et des données erronées seront transmises.

Dans le cadre de notre étude, nous avons simulé les communications bloquantes de la norme MPI telles qu'elles sont implémentées dans LAM/MPI [8][71]. Nous décrivons ici les protocoles de communication définis par l'implémentation LAM/MPI pour les communications bloquantes en utilisant TCP. LAM/MPI utilise trois protocoles différents pour implémenter la notion de communication bloquante : un protocole pour les messages longs et deux protocoles pour les messages courts (synchrone et asynchrone) [40].



(a) Messages longs (b) Messages courts synchrones (c) Messages courts asynchrones

Fig. 6.10 – Les protocoles de communication de LAM/MPI

- Messages longs (figure 6.10(a)) : l'échange d'un message long est réalisé en deux étapes. Dans un premier temps, seule l'enveloppe du message, contenant notamment la taille du message est envoyée au destinataire (1). Le destinataire compare alors la taille du message présente dans l'enveloppe à la taille de la zone tampon destinée à la réception du message. Un message d'accusé de réception est alors envoyé à l'émetteur (2). Ce dernier débute ensuite l'envoi du corps du message (3).
- Messages courts (figure 6.10(b)) : l'enveloppe et le corps du message sont envoyés en une seule communication (1). Le message est considéré traité dès la fin de l'envoi.
- Messages courts synchrones (figure 6.10(c)) : l'enveloppe et le corps du message sont envoyés en une seule communication (1). Le message est considéré comme traité par l'émetteur lorsqu'un accusé de réception provenant du destinataire a été reçu (2).

La norme MPI définit de manière générale la notion d'envoi bloquant, cependant cette notion de communication bloquante varie selon le protocole de communication utilisé et l'implémentation choisie. Afin de pouvoir simuler ce phénomène de blocage d'une tâche émettrice, l'étude de la Request Progression Interface (RPI) TCP de LAM/MPI est indispensable.

6.4.3.3 La RPI TCP de LAM/MPI

Afin de pallier les différences entre la vitesse de génération des messages par une application émettrice, la vitesse de transfert de ces messages par le protocole TCP/IP et la vitesse de lecture des messages par l'application réceptrice, le protocole TCP utilise une zone tampon d'émission et une zone tampon de réception (figure 6.11).

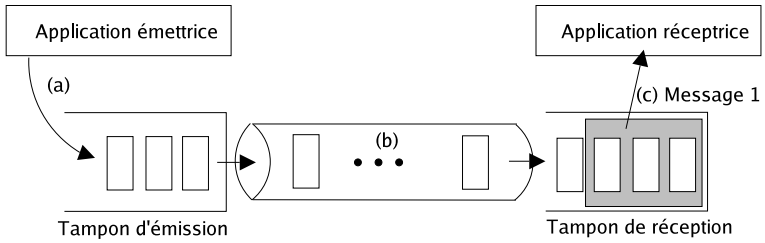


Fig. 6.11 – Les zones tampon de TCP

D'un point de vue de l'émetteur, les paquets correspondant aux messages générés par l'application émettrice sont stockés dans la zone tampon d'émission (a) et y résident jusqu'à la réception d'un accusé de réception. Du côté du récepteur, les paquets sont stockés dans la zone tampon de réception au rythme de l'algorithme TCP (b). Lors de la lecture d'un message par l'application réceptrice, la zone tampon de réception est vidée d'un nombre de paquets correspondant à la taille du message lu (c). La taille des zones tampon d'émission et de réception sont deux valeurs paramétrables par l'application.

Nous allons maintenant décrire l'utilisation faite par LAM/MPI de ces zones tampon TCP lors de l'émission et de la réception de messages[70] :

- Lors de l'émission d'un message, LAM/MPI tente d'écrire le message dans la zone tampon d'émission. Cette écriture est réalisée par un appel à la primitive système *write*. Dans le cas d'un envoi bloquant, la tâche émettrice reste bloquée jusqu'à ce que la totalité du message soit écrite dans la zone tampon d'émission. Si la zone tampon d'émission ne permet pas de contenir la totalité du message à envoyer, une partie seulement des données est écrite dans la zone tampon afin de remplir cette dernière, et LAM réalise une boucle active jusqu'à ce que la totalité du message soit transférée dans la zone tampon.
- La réception d'un message débute lorsqu'une requête de lecture (primitive MPI_Recv) est postée. LAM utilise la primitive système *read* et réalise une boucle active jusqu'à ce que la totalité du message soit lue dans la zone tampon de réception.

6.4.4 Caractéristiques des machines

Les performances d'une machine dépendent de plusieurs facteurs : l'architecture matérielle de la machine (inter-connexion et performances des bus de communication), les performances

intrinsèques de chaque composant de la machine et l'utilisation faite de ces composants par le système d'exploitation.

Les machines utilisées pour constituer des grappes de calcul possèdent des architectures matérielles et des composants semblables à ceux utilisés pour les machines grand public. La figure 6.12 donne un exemple type d'architecture pour ce genre de machines.

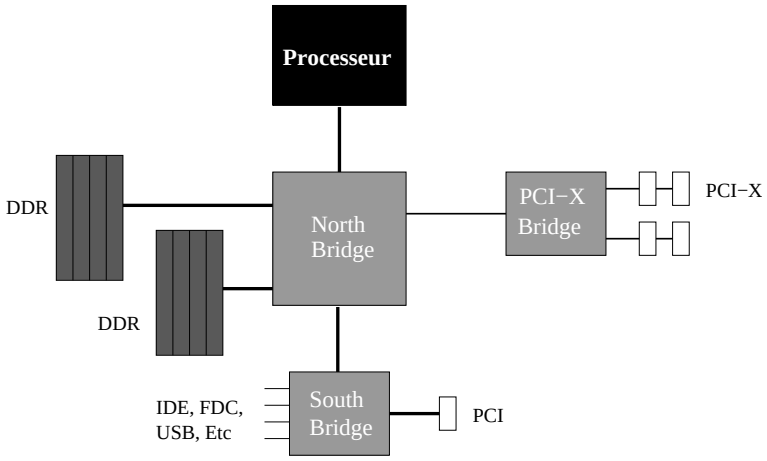


Fig. 6.12 – Architecture type d'un serveur

Les différents composants d'une machine que sont le processeur, le disque, et les périphériques partagent un accès commun à la mémoire vive par l'intermédiaire du *North Bridge*. De la même manière, les périphériques IDE, USB et PCI sont multiplexés sur un bus de communication commun grâce au *South Bridge*. Les échanges de données entre la mémoire, les disques et les périphériques peuvent être soit pilotés par le processeur, soit être réalisés parallèlement au travaux du processeur grâce aux contrôleurs DMA (*Direct Memory Access*).

Simuler le fonctionnement réel d'une machine, dans sa globalité, en tenant compte des échanges de données complexes intervenant sur les différents bus de communication, des interruptions matérielles générées par les différents périphériques et des politiques d'ordonnancement utilisées par les contrôleurs DMA est une opération complexe et coûteuse. De plus, le niveau de détails de nos modèles d'applications ne nous permettrait pas de mener à bien une telle simulation. De ce fait, nous avons choisi d'utiliser un modèle simplifié du fonctionnement d'un serveur, centré autour des quatre composants principaux ayant un impact sur les performances de nos applications : le(s) processeur(s), le(s) disque(s), l'interface réseau et la mémoire. Nous faisons donc l'hypothèse que les différents bus de communications sont dimensionnés de manière à ne pas représenter de goulots d'étranglements significatifs.

Le modèle proposé (figure 6.13), utilise des files d'attente pour représenter le processeur, le disque et les entrées/sorties réseau. La mémoire, quant à elle, est représentée par un compteur dont la valeur est bornée par la taille de la mémoire physique du serveur. Ce compteur est incrémenté ou décrémenté à chaque création ou destruction d'une instance d'application. La valeur utilisée pour cette incrémentation correspondant à une estimation de la taille mémoire utilisée par le processus associé.

Une file supplémentaire, appelée ordonnanceur permet d'aiguiller les requêtes de traitement vers la ressource souhaitée, en adéquation avec l'état d'avancement de l'application.

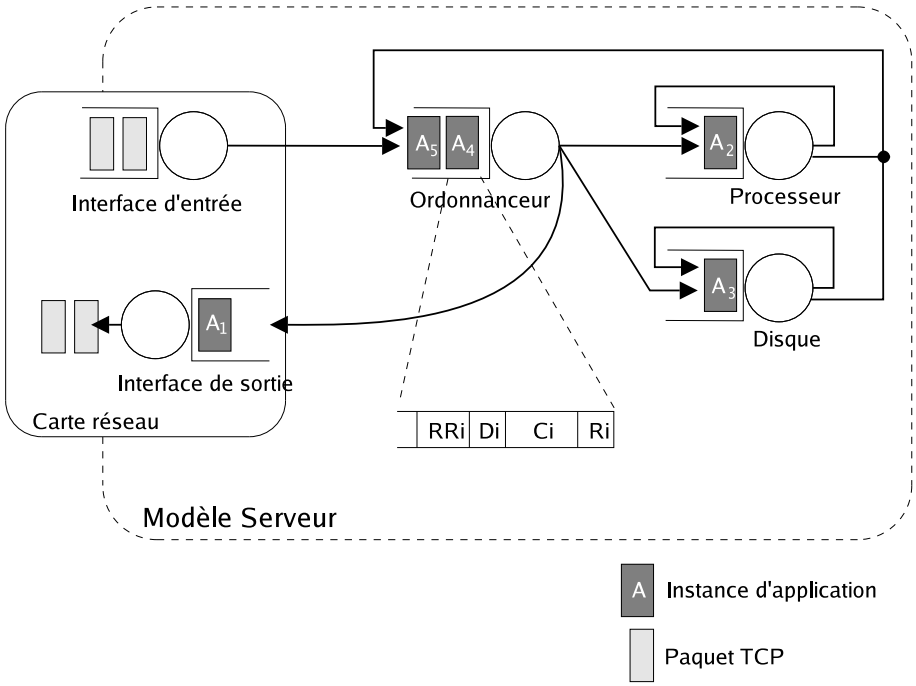


Fig. 6.13 – Modèle serveur

La création d'une requête applicative correspond à la réception d'un message sur l'interface d'entrée de la carte réseau. La requête ainsi créée est alors insérée dans la file de l'ordonnanceur. Ce dernier traite les requêtes une par une dans leur ordre d'arrivée et les aiguille vers la file d'attente permettant de réaliser la phase de traitement courante : la file processeur pour une phase de calcul, la file disque pour une phase d'accès au disque, l'interface de sortie de la carte réseau pour l'envoie d'un message.

Chaque file d'attente possède son propre mode de gestion des requêtes :

– La file disque traite ses requêtes en mode *Round Robin*. La taille du quantum alloué à chaque requête est un multiple de la taille d'une page mémoire. Ce mode de traitement permet de simuler de manière simplifiée le partage de l'accès au disque par des contrôleurs DMA. Dans la réalité, les DMA des disques durs utilisent des politiques complexes de sélection de l'ordre de traitement des requêtes, dans le but d'optimiser la vitesse de transfert des informations (priorité aux données situées dans la mémoire cache du disque, optimisation des déplacements de la tête de lecture ...). Afin de pouvoir simuler des disques ayant des performances diverses, deux paramètres sont utilisées pour exprimer le taux de service de la file disque. Le premier paramètre correspond à la taille (en octets) d'un quantum disque. A chaque terminaison d'un quantum, la taille de la phase d'accès au disque de l'application est décrémentée de cette valeur. Le deuxième paramètre correspond au temps nécessaire au traitement des données d'un quantum, c'est à dire au débit du disque. En modifiant la valeur de ce deuxième paramètre, il est ainsi possible de simuler le fonctionnement de disques plus ou moins rapides. Lorsqu'une requête a terminé sa phase d'accès au disque, elle est ré-insérée dans la file ordonnanceur,

en dernière position.
- La file processeur traite également les requêtes en mode *Round Robin* de manière analogue à l'ordonnanceur du système d'exploitation (cf section 6.4.5.1). Comme pour la file disque, deux paramètres permettent de définir le taux de service de la file processeur. Le premier spécifie la taille d'un quantum processeur (en ms), le deuxième permet de spécifier la durée de traitement d'un quantum sur un processeur donné. Lorsqu'une requête a terminé sont traitement de calcul, elle est ré-insérée dans la file ordonnanceur, en dernière position.

6.4.5 Comportement du système d'exploitation

Dans le cadre de notre étude, trois composants du système d'exploitation peuvent influencer les performances du système modélisé : le système d'ordonnancement des tâches, le mécanisme de SWAP et la gestion de l'interaction avec la carte réseau. Nous avons retenu les systèmes d'exploitation de type Unix et plus particulièrement le noyau 2.6 de linux.

6.4.5.1 L'ordonnancement des tâches

Le noyau 2.6 de linux définit trois politiques d'ordonnancements distinctes :
- deux politiques pour les tâches temps réels (First In First Out et Round Robin)
- une politique pour les tâches utilisateurs dites "normales"

Seule cette dernière politique d'ordonnancement en temps partagé (politique par défaut) est considérée dans le cadre de notre étude. Une file d'exécution (*runqueue*) est associée à chaque processeur du système. Le rôle de cette structure de données est de gérer l'ensemble des tâches éligibles sur le processeur. Une *runqueue* gère deux tableaux de tâches (nommés "active" et "expired") indexés par des niveaux de priorité (figure 6.14). 140 niveaux de priorité sont gérés (plus le niveau est faible, plus la tâche est prioritaire) mais seuls les niveaux compris entre 100 et 139 sont associables à des tâches utilisateurs, les autres niveaux étant réservées aux tâches systèmes.

Lors de sa création, chaque tâche se voit associer une valeur *nice* (toujours comprise entre -20 et 19), l'ordonnanceur calcule alors la priorité de la tâche (120 + *nice*) et la place dans le tableau des tâches "active".

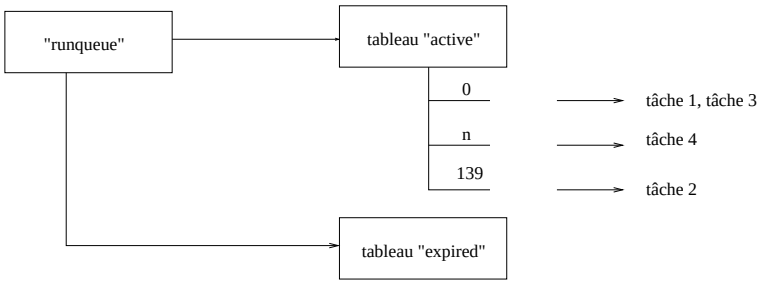


Fig. 6.14 – L'ordonnanceur de Linux 2.6

L'ordonnanceur sélectionne dans le tableau "active" la tâche de priorité maximale (valeur la plus faible). Il calcule son quantum de temps nommé *timeslice* puis l'exécute. Dans le cas où plusieurs tâches de même priorité sont présentes, l'ordonnanceur effectue un traitement

Round Robin entre ces dernières. Lorsque le timeslice d'une tâche est épuisé, la (ou les) tâche(s) exécutée(s) sont placées dans le tableau "expired", leur futur *timeslice* est calculé, puis l'ordonnanceur passe au niveau de priorité suivant. Une fois toutes les tâches du tableau "active" traitées, le contenu du tableau "expired" est transvasé dans le tableau "active".

6.4.5.2 Le mécanisme de SWAP

Le mécanisme de SWAP peut intervenir de deux manières différentes :

- Lors de la création d'une instance d'application, si l'image mémoire du processus n'est pas déjà en mémoire, un accès disque d'une taille équivalente à la taille de l'image mémoire du processus associé est réalisé afin de charger l'image du processus en mémoire. Si la quantité de mémoire libre est insuffisante pour accueillir le processus en question, un certain nombre de pages mémoire doit être déchargé de la mémoire vers le disque afin de permettre l'exécution de la nouvelle application.
- Lorsqu'un processus est élu par l'ordonnanceur pour accéder au processeur, il se peut qu'une partie de l'espace mémoire alloué à ce processus ait été déchargée de la mémoire par le mécanisme de SWAP. Dans ce cas, les pages mémoire associées doivent être à nouveau chargées en mémoire, entraînant éventuellement le déchargement de nouvelles pages appartenant à une application différente. La probabilité qu'un processus en cours d'exécution ait été déchargé de la mémoire physique est d'autant plus grande que le nombre d'applications présentes en SWAP est important. Dans le cadre du simulateur, cette probabilité est calculée de la manière suivante :

$$\frac{\text{Mémoire SWAP utilisée}}{\text{Mémoire totale utilisée (SWAP + RAM)}} \tag{6.1}$$

6.4.5.3 Gestion des entrées/sorties réseau

Un des rôles du système d'exploitation est de réaliser le lien entre les protocoles applicatifs (couche 7 du modèle OSI, voir figure 6.15) et la carte réseau (couche 1 du modèle OSI). Ce lien est réalisé conjointement par les drivers de la carte réseau, et dans le cas de l'utilisation du protocole TCP/IP, par la pile TCP/IP (couches liaison, réseau et transport du modèle OSI).

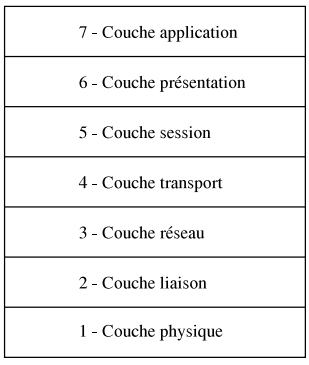


Fig. 6.15 – Modèle OSI

L'implémentation Linux de la pile TCP/IP [24] est intégrée au noyau du système d'exploitation. Nous proposons d'étudier le comportement du noyau 2.6 de Linux lors de la réception et de l'envoi de messages :

- Réception d'un message : lorsque des paquets ethernet sont reçus par la carte réseau, cette dernière génère une interruption matérielle afin d'informer le système d'exploitation de la présence de données à traiter. Le *handler* d'interruption suspend l'exécution de toutes les tâches du système, de ce fait son traitement doit être le plus bref possible afin de ne pas créer un phénomène de famine auprès des processus utilisateurs. Le rôle du *handler* d'interruption se limite ainsi à stocker les données reçues dans une zone mémoire du noyau appelée *queuing interface*.
  Une *softIRQ* [81] dédiée au traitement des données réseaux reçues traite par la suite les données présentes dans la *queuing interface*, conformément à l'algorithme de TCP/IP, lorsqu'elle accède au processeur.
- Émission de paquets : lors de l'émission d'un message, les données à transmettre sont placées dans la *queuing interface* puis prises en charge par une *softIRQ* dédiée à l'émission. Cette dernière gère le contrôle de flux (fragmentation, construction des entêtes IP ...), place les paquets à envoyer dans une file locale de l'espace mémoire du driver de la carte réseau, et déclenche une interruption.
  La carte réseau prend ensuite en charge le transfert des paquets présents dans le driver.

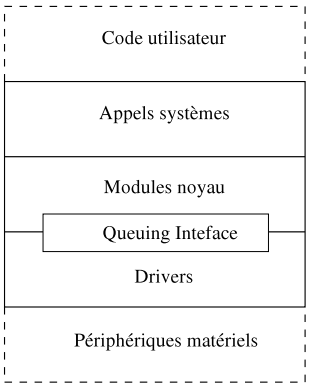


Fig. 6.16 – Architecture en couche de Linux

Malgré l'utilisation de *softIRQ* afin de limiter l'impact des interruptions sur les performances des applications, le mécanisme d'interruption peut conduire à une saturation de la machine dans le cas d'utilisation de réseaux haut débit (1Gb/s - 10 Gb/s). Pour palier ce problème, deux solutions sont proposées à la fois par les constructeurs de cartes réseau et par les concepteurs de systèmes d'exploitation [81] :

- Certaines cartes réseau implémentent la notion d'*interrupt mitigation*. Au lieu de générer une interruption à chaque réception d'un nouveau paquet, les paquets sont stockés dans une mémoire tampon intégrée à la carte réseau. Un interruption est alors générée, soit lorsqu'un certain nombre de paquets a été stocké, soit lors de l'expiration d'un délai.
- La version 2.6 du noyau Linux utilise également un mécanisme dénommé NAPI (New API) afin de limiter l'utilisation des interruptions. Lors de la réception d'une interruption indiquant la présence de nouvelles données en réception, le noyau désactive les interruptions. Par la suite, lorsque les nouvelles données auront été traitées par la *softIRQ* de réception, cette dernière sondera le driver de la carte réseau afin de récupérer, le cas échéant de nouvelles données à traiter. Si aucune donnée n'est disponible, le mécanisme d'interruption est réactivé. De cette manière, si les paquets arrivent plus rapidement sur le réseau qu'ils ne peuvent être traités par le système d'exploitation, aucune interruption

n'est générée et les paquets sont lus à la vitesse du noyau.

6.4.6 Le nœud Ordonnancement

Nous nous intéressons, ici à l'ordonnancement au niveau du frontal du système ASP. Le rôle de ce frontal est de répartir les requêtes des clients sur les différentes machines de la grappe. Dans un premier temps, seules deux politiques d'ordonnancement sont considérées :

- Politique de partage de charge : l'ordonnanceur choisit un serveur différent à chaque nouvelle requête. Une liste circulaire de tous les serveurs de la grappe est utilisée.
- Politique d'équilibrage de charge : l'ordonnanceur choisit le serveur traitant le moins de requêtes à la date d'arrivée de la nouvelle requête.

6.5 Intégration au sein du simulateur DHS

Les modifications apportées au simulateur DHS afin de simuler le système décrit précédemment vont maintenant être présentées.

6.5.1 La couche réseau

Le simulateur DHS permet de simuler le fonctionnement du protocole de communication TCP version NewReno [35], sans se soucier des informations contenues dans les paquets transportés et des applications utilisant ces paquets. Le simulateur a été enrichi afin de simuler les zones tampon d'émission et de réception du protocole TCP (annexe B).

Un source TCP doit être associée à chaque couple de tâche devant échanger des messages. Cette source gère l'ensemble des communications entre les deux tâches de la même application. Cette source TCP a été modifiée afin de pouvoir simuler le fonctionnement d'un protocole applicatif. Elle gère la liste des messages en cours de transfert sur le réseau, et positionne les paramètres du protocole TCP (taille des zones tampon, *Maximum Segment Size*, valeur des différents délais de TCP, mécanisme de *nagle* (annexe B)...) conformément aux valeurs utilisées par le protocole applicatif simulé.

6.5.2 Les clients

Un client est représenté par un nœud client et une source de trafic (figure 6.17) :

- Le nœud client représente un emplacement du réseau à partir duquel un ou plusieurs clients soumettent des requêtes. Il est composé d'une interface d'entrée et d'une interface de sortie.
- La source de trafic génère des événements conformément à la distribution d'inter-arrivée entre deux requêtes. Chaque événement représente une requête de traitement d'un même client vers la même application. La taille de la requête associée à chaque événement suit une distribution donnée en fonction des caractéristiques du client et du type d'application demandée.

6.5.3 L'application séquentielle

Une application séquentielle, un serveur HTTP ou une tâche d'une application est représentée par un générateur d'application et par un paquet application.

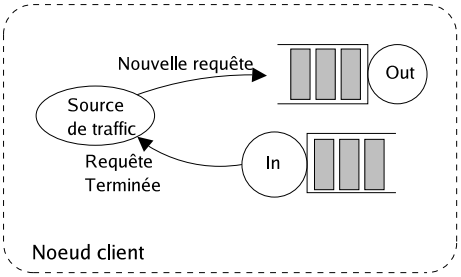


Fig. 6.17 – Le nœud client

Le générateur d'application regroupe l'ensemble des caractéristique communes à toutes les instances d'une application particulière :
- la distribution de la taille des requêtes générées par les clients de l'application,
- la distribution de la taille de la réponse de l'application à une requête d'un client,
- la distribution de la ou des phases d'accès au disque,
- la distribution de la ou des phases de calcul,
- la distribution de la taille mémoire consommée par une instance de l'application
- des paramètres spécifiques à l'application : par exemple les valeurs `MaxClients`, `MaxRequestsPerChild`, `KeepAlive` et `KeepAliveTimeout` du serveur HTTP Apache.

Le générateur d'application détermine également l'enchaînement des différentes phases de traitement de l'application. Cette enchaînement est déterminé par un automate. A chaque terminaison d'une phase de traitement, la phase suivante est déterminée en tenant compte de la nature de la phase écoulée. Différents états sont définis afin de déterminer les phases de traitement d'une application séquentielle :
- `UNDEFINED` : l'application n'a pas débuté son exécution,
- `IN` : le paquet doit réaliser une opération de lecture sur le réseau,
- `OUT` : le paquet doit réaliser une opération d'envoi sur le réseau,
- `LOAD` : l'image mémoire de l'application doit être chargée en mémoire,
- `DISK` : l'application réalise un accès disque,
- `CPU` : l'application réalise une opération de calcul,
- `SWAP` : la mémoire est saturée. Une partie de la mémoire doit être libérée, afin de rendre possible l'exécution de l'application,
- `END` : l'exécution de l'application est terminée.

Un exemple d'automate déterminant l'enchaînement des phases d'accès aux ressources d'une application séquentielle est donné par la figure 6.18.

Le paquet application représente l'état d'une instance d'application en cours d'exécution. Il contient les informations associées à la requête :
- identifiant de la requête,
- taille de la requête,
- taille de l'image mémoire utilisée par la requête,
- taille de la ou des phase(s) de calcul,
- taille de la ou des phase(s) d'accès au disque,

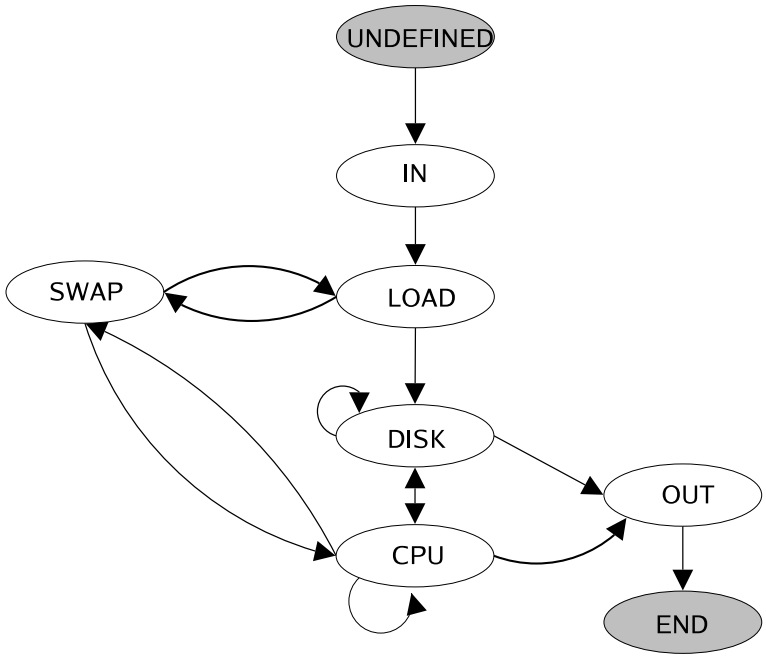


Fig. 6.18 – Enchaînement des différentes phases de traitement d'une application séquentielle

- taille de la réponse,
- état courant de la requête.

Chacune des tailles est renseignée en utilisant les générateurs aléatoires du générateur d'application.

6.5.4 Les applications parallèles

Les applications parallèles sont représentées par un gestionnaire d'application contenant, soit le graphe de l'application parallèle, soit le schéma de l'application maître/esclaves. Ce gestionnaire d'application représente l'état de l'application parallèle : tâches terminées, tâches en cours d'exécution, machines sur lesquelles chaque tâche parallèle est placée.

Chaque tâche de l'application parallèle est représentée par un paquet application. Lors de la création d'une application parallèle, une requête de création de tâche est envoyée sur chaque machine devant traiter une tâche de l'application. Une telle requête engendre la création d'un paquet application sur la machine associée. La tâche créée débute alors son exécution jusqu'à ce qu'elle soit bloquée en attente d'un message. Lorsqu'un message est reçu, le paquet application est réveillé et le message est alors traité (lecture du message, et réalisation du traitement associé).

Lors de la réception d'un message synchrone, le traitement associé correspond à l'envoi d'un message d'accusé de réception à la tâche émettrice. Si tous les messages en attentes ont été reçus, le paquet application démarre une phase de traitement (phases de calcul et de disque).

Lors de l'envoi d'un message, le message transite sur la file de sortie, puis est pris en charge par la source TCP. Dans le cas d'un envoi synchronisé, le paquet application est bloqué en attente d'un accusé de réception, dans le cas d'un envoi asynchrone, l'application poursuit son exécution (envoi du message suivant ou mise en attente).

6.5.5 Le nœud serveur

Une machine serveur est constituée d'une interface d'entrée, d'une interface de sortie, d'une file ordonnanceur et de files round-robin pour le disque et le processeur (figure 6.19). L'interface d'entrée est destinée à recevoir des paquets TCP provenant d'un client ou d'une application située sur une machine différente. Dès que tous les paquets TCP associés à une requête ont été traités par l'interface d'entrée, un paquet application représentant la requête est créé. Ce paquet application est alors injecté dans la file ordonnanceur, puis dans les différentes files round-robin. La sélection de la file dans laquelle est insérée le paquet application est réalisé par le mécanisme de routage, en utilisant les informations renvoyées par le générateur d'application. Une fois le traitement terminé, le paquet application est injecté dans l'interface de sortie, qui génère les paquets TCP correspondant à la réponse du serveur.

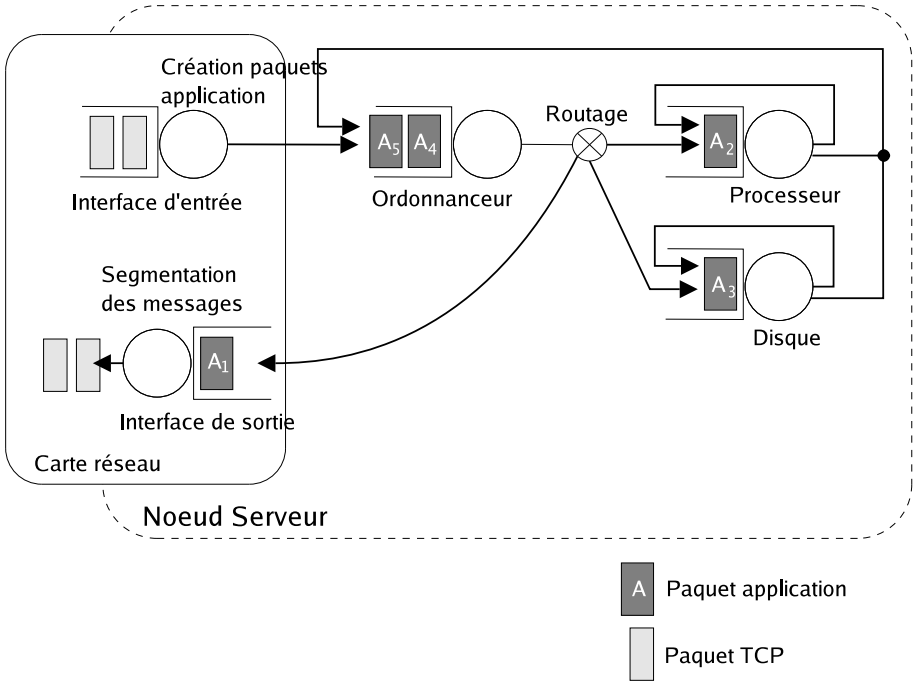


Fig. 6.19 – nœud serveur

Les interfaces d'entrée et de sortie traitent les paquets TCP à la vitesse de la carte réseau (10/100 Mb/s...). La file ordonnanceur route les paquets application vers le composant adéquat, en fonction l'état dans lequel se trouve le paquet application :

- état `DISK` → file disque,
- état `LOAD` → file disque,
- état `SWAP` → file disque,
- état `CPU` → file processeur,
- état `IN` → interface d'entrée,
- état `OUT` → interface de sortie,
- état `END` → interface de sortie.

6.5.6 Le nœud ordonnanceur

Tout comme le nœud client, le nœud ordonnanceur est composé d'une interface d'entrée et d'une interface de sortie. Ce nœud sert de routeur entre les clients et les serveurs. Tous les paquets TCP d'une même requête sont routées vers le même serveur, choisi en fonction d'un algorithme d'ordonnancement particulier.

6.5.7 Le système global

La figure 6.20 représente le système simulé dans sa globalité.

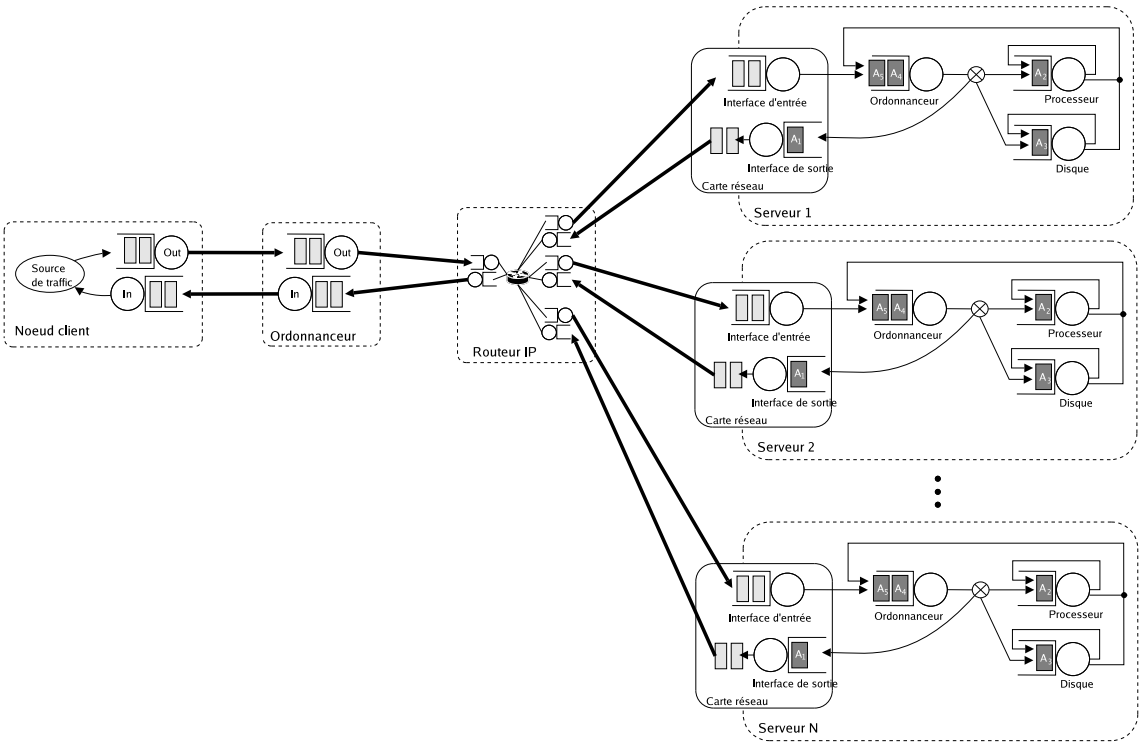


Fig. 6.20 – Système global

Le nœud "Routeur IP" permet de simuler le fonctionnement d'un routeur ou d'un switch inter-connectant les différentes machines de la grappe. Chaque port du routeur réel est représenté par une file d'entrée et une file de sortie. Il est possible de limiter le nombre de paquets pouvant être mis en attente sur chaque file de sortie du routeur afin de simuler les files d'attente d'un routeur réel. Le simulateur DHS permet également de remplacer ce routeur IP par un nuage Internet qui représente des connexions longue distance via des opérateurs de communication.

Chaque flèche en gras sur la figure 6.20 représente un lien physique entre deux cartes réseau ou une carte réseau et un port du routeur. Afin de prendre en compte les délais engendrés par la transmission d'un paquet sur un lien physique, un délai est inséré sur chaque interface de sortie. Lorsqu'un paquet quitte une interface de sortie, il est retardé d'une valeur égale à ce délai.

6.5.8 Indicateurs de performance

Un nombre important d'indicateurs de performances peuvent être fournis par le simulateur. Ces indicateurs fournissent des informations plus ou moins pertinentes en fonction du cadre d'utilisation souhaité. Nous allons présenter les principaux indicateurs utilisés dans le cadre de notre étude, et leur utilité vis à vis du dimensionnement du système.
Pour chaque application, les principales valeurs statistiques utilisées sont les suivantes :

- le temps de réponse moyen sur chaque serveur de la grappe : ce dernier est donné par le délai de bout en bout du flux application de chaque serveur,
- l'espérance du nombre de requêtes présentes dans le système,
- le temps de réponse de bout en bout : ce dernier est calculé en sommant les délais de bout en bout de chaque flux TCP et la moyenne des temps de réponse sur chaque serveur.

Les deux premiers indicateurs fournissent une indication sur la charge de travail imposée par l'application à chaque machine de la grappe. Cette information permet de détecter d'éventuels déséquilibres entre les charges des différents serveur de la grappe, et de modifier la politique d'ordonnancement en conséquence.

Le troisième indicateur, le temps de réponse de bout en bout d'une application, permet de caractériser la qualité du système mis en place. Cette valeur correspond au délai entre la soumission d'une requête par un client et l'obtention de sa réponse. Elle représente, de ce fait, la qualité du système perçue par un client. Cette valeur doit être comparée au temps d'exécution de la même requête sur un système vide, d'une part, et au temps de réponse espéré par le client, d'autre part, afin de déterminer si un délai trop long est dû à une mauvaise utilisation des ressources ou à un sous-dimensionnement du système.

Pour chaque serveur de la grappe de machine, des informations sur l'utilisation de chaque composant (interface réseau, disque, processeur) sont également disponibles. Ces indicateurs permettent de détecter si un de ces composants représente un goulot d'étranglement.

Enfin, pour chaque flux TCP, les informations statistiques concernant le délai de bout en bout, la gigue ou encore les pertes sont visualisables. Ces informations permettent de détecter si le réseau représente un facteur limitant de la performance de la grappe de machine.

6.6 Conclusion

Les différentes techniques couramment utilisées afin d'estimer les performances d'applications ont été présentées. Chacune de ces techniques possède ses propres avantages et inconvénients.

Les benchmarks permettent d'obtenir une information exacte sur le comportement d'une application. Cependant, les systèmes étudiées par cette technique se limitent généralement à l'étude d'une seule application, s'exécutant sur un nombre de machines réduit. En effet, leur mise en œuvre sur des systèmes plus complexes peut s'avérer délicate et longue. Obtenir des indicateurs de performances pertinents d'un benchmark est également une tâche délicate qui nécessite d'instrumenter une application existante, ou de se contenter d'une vision boîte noire du système qui n'offre que peu d'information sur les facteurs limitant la performance du système global. De plus, le spectre des systèmes étudiés se limite aux systèmes déjà existants.

La simulation permet de pallier certains inconvénients des benchmarks : elle ne se limite pas qu'aux systèmes existants et en fonction du niveau de détail choisi, le délai d'obtention d'un résultat peut être très bref. Les indicateurs retenus pour étudier le problème de performance

sont également illimités. Cependant, la qualité d'une simulation dépend du rapport entre la précision des résultats souhaités et le temps d'obtention de ces résultats. De plus, bâtir un modèle détaillé d'un système complexe est une tâche délicate qui peut constituer un frein au choix de la simulation.

La modélisation analytique permet d'obtenir des résultats d'assez bonne précision (selon le système étudié) de manière très rapide. Cependant, le nombre de système entièrement modélisables analytiquement reste faible.

Les caractéristiques du système étudié ont ensuite été détaillées. Les modèles de simulation mis au point pour étudier ce système, ainsi que leur intégration au sein du simulateur DHS ont finalement été exposés. Les résultats fournis par ces différents modèles, ainsi que leurs performances vont maintenant être présentés.

Chapitre 7

Validation.

Sommaire

7.1 Introduction

Le but de ce chapitre est de présenter les différents tests effectués afin de valider et d'évaluer les performances du simulateur d'applications présenté précédemment.

Deux types d'études sont considérées. La première, qui concerne les applications interactives, compare les résultats obtenus par simulation à des résultats analytiques obtenues en utilisant des modèles de réseaux de files d'attentes. Le but de cette étude est de valider les modèles de simulation client, application et machine utilisés, et d'étudier le comportement du modèle simplifié de serveur Web, en confrontant les résultats obtenus à des résultats analytiques connus.

La deuxième étude, concerne quant à elle les applications parallèles. Elle compare les durées réelles d'exécution d'une application parallèle de référence à l'estimation faite par le simulateur de ces mêmes durées. L'application parallèle utilisée pour cette étude et un *benchmark* permettant de faire varier le grain de calcul. L'objectif de cette deuxième étude est d'étudier la précision et les performances de différents modèles d'applications parallèles et de communication réseau.

7.2 Performances de serveurs Web

Cette étude vise à prédire le temps d'obtention d'une page Web par un client. Elle compare les résultats obtenus en utilisant le modèle de simulation de serveur Web présenté précédemment et des résultats analytiques obtenus en étudiant un réseau de files d'attentes modélisant le même serveur Web.

Dans un premier temps, le système étudié se compose d'un unique serveur Web, répondant aux requêtes de plusieurs clients issus d'une même source de trafic applicatif. Le comportement du serveur est étudié avec différentes charges de travail, jusqu'à saturation de ce dernier.

Ensuite, la même étude est réalisée en considérant des clients ayant des comportements et accédant à des documents ayant des caractéristiques différentes.

Enfin, l'effet de l'ajout de plusieurs serveurs, se partageant la charge de travail est simulé.

7.2.1 Serveur unique et un seul type d'application

Cette étude concerne le comportement d'une unique machine hébergeant un serveur Web. Tous les clients se connectant au serveur ont le même comportement et accèdent au même type de contenu. L'objectif de cette étude est de déterminer le nombre de clients maximal pouvant être traité dans des temps acceptables par une machine donnée. Le système simulé est composé d'une unique machine, reliée directement à une source de clients par deux liens réseau (figure 7.1).

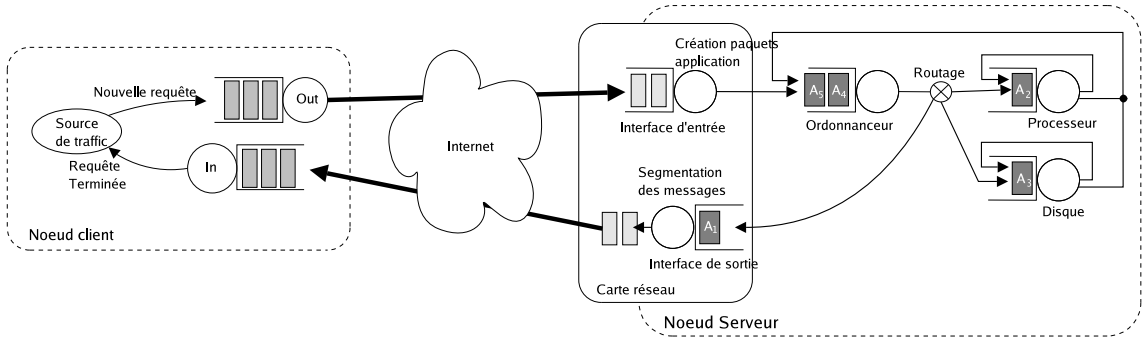


Fig. 7.1 – Simulation d'un unique serveur Web

7.2.1.1 Modèle analytique du système étudié

Le réseau de files d'attentes utilisé pour modéliser le serveur Web est représenté par la figure 7.2. La machine est représentée par trois files *round robin*, pour le processeur, le disque et l'interface réseau. Le taux de service μ_i de chacune de ces files est un paramètre de l'application. L'arrivée de nouveaux clients dans le système est modélisé par le taux d'arrivées de sessions $\Lambda_{session}$. Au cours d'une session, un client réalise un nombre de requêtes représenté par une variable aléatoire de moyenne N_r. Le temps de réflexion entre deux requêtes, qui correspond au temps de lecture de la page par le client, est modélisé par un délai appliqué à chaque requête.

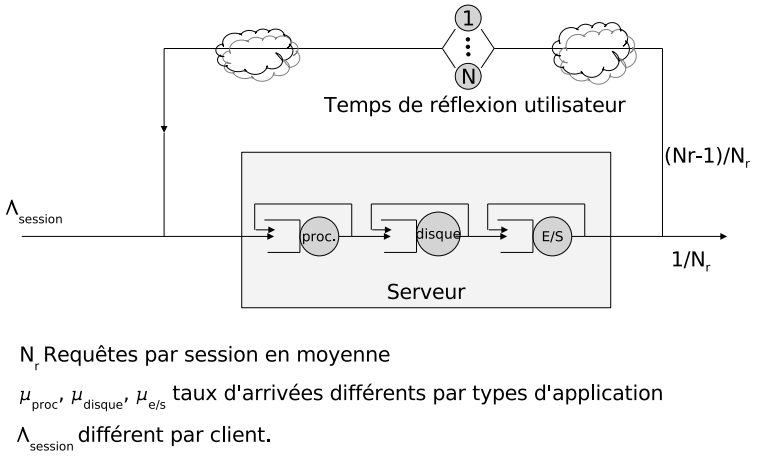


Fig. 7.2 – Réseau de files d'attentes d'un serveur Web

Le temps de réponse du serveur Web à une requête d'un client noté $R_{serveur}$ correspond à la somme des temps passée par la requête dans chacune des files "processeur", "disque" et "entrées/sorties" :

$$R_{serveur} = R_{proc} + R_{disque} + R_{E/S}. \tag{7.1}$$

On note D_{proc}, $D_{E/S}$, D_{disque} la charge de travail demandée par le traitement d'une requête de l'application considérée à la file processeur, respectivement disque et entrées/sorties. On note $D_{chargement}$ la charge de travail imposée au disque lors du chargement en mémoire d'un nouveau processus.

La valeur D_{proc} correspond directement au temps processeur utilisé par une requête du client, les autres valeurs sont calculés à partir des relations suivantes :

$$D_{disque} = \frac{\text{taille de la page Web}}{\text{taux de transfert du disque}} \tag{7.2}$$

$$D_{E/S} = \frac{\text{taille de la page Web}}{\text{débit de l'interface réseau}} \tag{7.3}$$

$$D_{chargement} = \frac{\text{taille image du processus}}{\text{taux de transfert du disque}} \tag{7.4}$$

Les taux de services de chaque file d'attente sont alors obtenus de la manière suivante :

$$\mu_{proc} = \frac{1}{D_{proc}}, \quad \mu_{E/S} = \frac{1}{D_{E/S}}, \quad \mu_{disque} = \frac{1}{D_{disque} + D_{chargement}} \tag{7.5}$$

Notons λ_r le taux d'arrivée des requêtes sur le serveur Web. Ce dernier est obtenu en faisant le produit du taux d'arrivée de sessions dans le système et du nombre moyen de requêtes par session :

$$\lambda_r = \Lambda_{session} \times N_r \tag{7.6}$$

Si l'on note Q_{proc}, Q_{disque}, $Q_{E/S}$ et $E(X)$ l'espérance du nombre de clients présents dans la file processeur, respectivement la file disque, la file d'entrées/sorties et le système global, on obtient alors les relations suivantes :

$$R_{proc} = \frac{Q_{proc}}{\lambda_r}, \quad R_{disque} = \frac{Q_{disque}}{\lambda_r}, \quad R_{E/S} = \frac{Q_{E/S}}{\lambda_r} \tag{7.7}$$

$$E(X) = Q_{proc} + Q_{disque} + Q_{E/S} \tag{7.8}$$

avec :

$$Q_{proc} = \frac{U_{proc}}{1 - U_{proc}}, \quad Q_{disque} = \frac{U_{disque}}{1 - U_{disque}}, \quad Q_{E/S} = \frac{U_{E/S}}{1 - U_{E/S}} \tag{7.9}$$

$$U_{proc} = \frac{\lambda_r}{\mu_{proc}}, \quad U_{disque} = \frac{\lambda_r}{\mu_{disque}}, \quad U_{E/S} = \frac{\lambda_r}{\mu_{E/S}} \tag{7.10}$$

où : U_{proc}, U_{disque} et $U_{E/S}$ représentent le taux d'utilisation de la file processeur, respectivement disque et entrées/sorties.

La valeur du paramètre `MaxRequestsPerChild` (voir section 6.4.2.2) peut être prise en compte dans le calcul du paramètre $D_{chargement}$, dans ce cas :

$$D_{chargement} = \frac{\text{taille image du processus}}{\texttt{MaxRequestsPerChild} \times \text{taux de transfert du disque}} \tag{7.11}$$

7.2.1.2 Caractéristiques du test

De nombreuses études destinées à caractériser le trafic HTTP sont disponibles dans la littérature [22][2][57][80][50][53]. Ces différentes études utilisent différentes loi de probabilités pour caractériser le trafic HTTP, selon le modèle de comportement utilisé (prise en compte ou pas des phases de réflexion du client, par exemple) et le type de contenu proposé par les serveurs Web (contenu plus ou moins riche en objets multimédias).

Le test présenté ici utilise des modèles issues d'études métrologiques de serveurs Web existants. Le modèle de comportement des clients utilisé pour la simulation est basé sur une étude métrologique d'un serveur Web réel de l'INRIA, présentée dans la thèse de Nicolas NICLAUSSE [53]. Le modèle application du serveur Web utilisé est paramétré en utilisant les paramètres par défaut du serveur Apache, et des valeurs de consommation de ressources mesurées sur une machine de référence.

L'ensemble des paramètres d'expérimentation sont résumés dans le tableau 7.1.

Les caractéristiques physiques de la machine de référence utilisée pour évaluer la consommation de ressources du serveur Apache sont les suivantes :

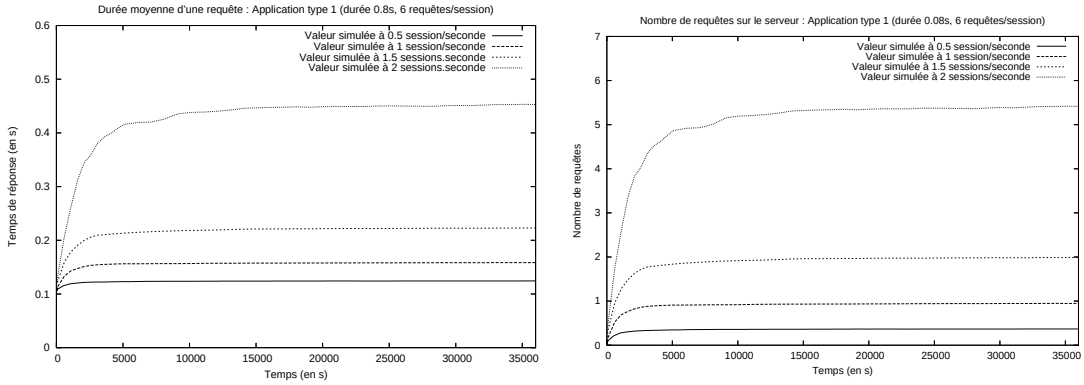
Processeur	pentium III 333 Mhz
Mémoire	128 Mo
Débit disque	20 Mo/s

Paramètre	distribution	caractéristiques
Temps processeur	déterministe	0.07 s
Taille image mémoire	déterministe	3 Mo
Taille d'une requête	lognormale	moyenne 360 octets, variance 106,5.
Taille des pages	lognormale	moyenne 19 Ko, variance 15376 Ko
Arrivées de sessions	exponentielle	
Nombre de requêtes par session	exponentielle arrondie entre 1 et 200	moyenne 6
Période de réflexion	lognormale	moyenne 68 s, variance 14641

Tab. 7.1 – Les différentes distributions utilisées

7.2.1.3 Temps de réponse du serveur

La figure 7.3 présente l'évolution du temps de réponse de la machine et du nombre de requêtes présentes dans le système au cours du temps, pour différentes valeurs d'inter-arrivées de sessions.

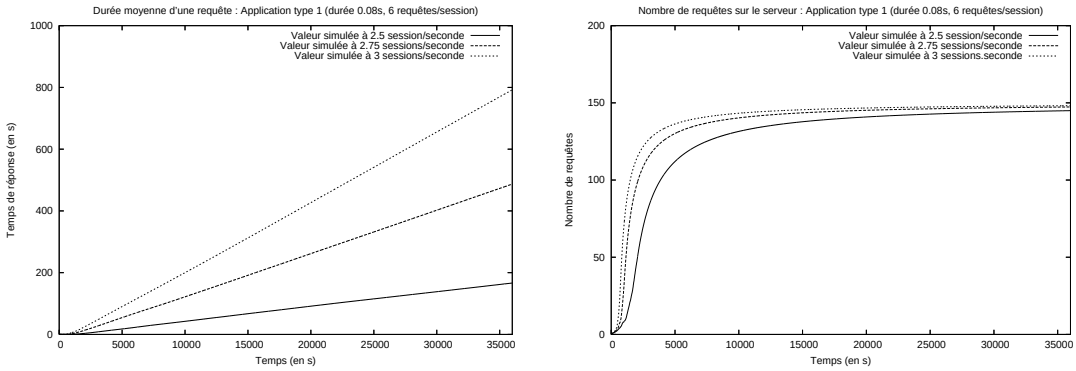


(a) Temps de réponse de la machine

(b) Nombre de requêtes dans le système

Fig. 7.3 – Évolution du système au cours du temps, à différentes charges

Le temps de traitement, sans tenir compte d'aucune contention pour une requête de l'application est de 0.08 secondes. Si le nombre de requêtes à traiter continue à augmenter, le serveur arrive à saturation, et le temps de réponse croit linéairement (figure 7.4). Ce cas de figure, où le temps de réponse croît indéfiniment n'est jamais atteint dans la réalité. En effet, lorsque le serveur est trop chargé, et que le temps de réponse devient inacceptable pour le client, ce dernier annule le chargement de sa page. La demande de traitement envers le serveur Web est de ce fait modifiée. Face à un temps de réponse trop long, le client peut également multiplier le nombre de requêtes (en réalisant plusieurs clics, dans l'espoir d'obtenir une réponse plus rapidement), ce qui a pour effet de saturer un peu plus le serveur.



(a) Temps de réponse de la machine (b) Nombre de requêtes dans le système

Fig. 7.4 – Évolution du système saturé

7.2.1.4 Comparaison avec le modèle analytique

Nous comparons ici les valeurs obtenues en régime stationnaire par simulation, à celles calculées analytiquement. Seule les valeurs pour lesquelles le système n'est pas saturé sont prises en compte. La figure 7.5 présente le pourcentage d'erreur entre les valeurs analytiques et simulées.

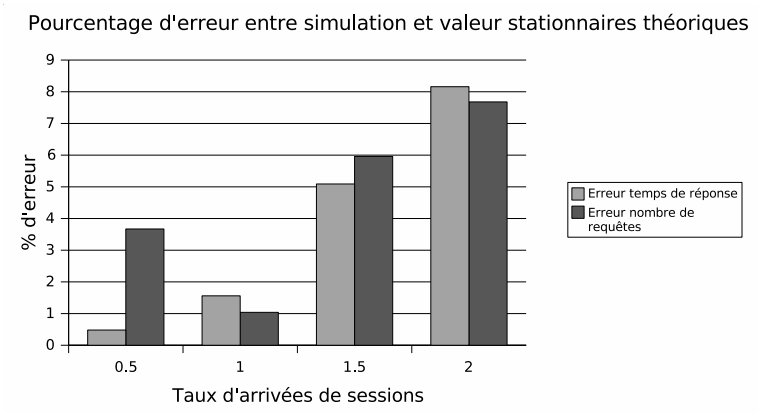


Fig. 7.5 – Comparaison entre résultats simulés et résultats analytiques

Les résultats obtenus par simulation sont bien conformes aux valeurs analytiques attendues. Les faibles pourcentages d'erreur obtenus peuvent être imputés à l'utilisation de distributions arrondies pour représenter le nombre de requêtes par session N_r réalisé par un client. Cette valeur est obtenue en simulation en arrondissant la valeur décimale obtenue par tirage aléatoire à la valeur entière la plus proche. De ce fait, la moyenne N_r obtenue peut différer de quelque centièmes de la valeur moyenne souhaitée.

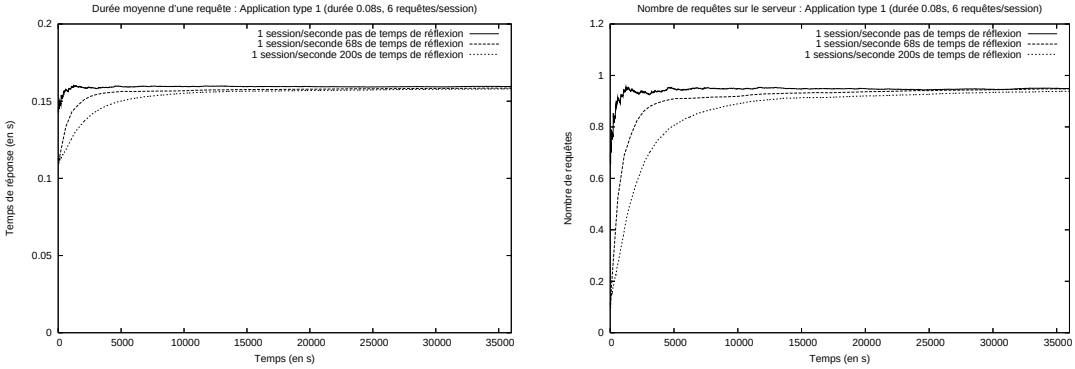
Les temps de réponse considérés pour l'application (0.08 s) étant très faibles, une faible erreur sur le taux d'arrivée des requêtes dans le système (qui dépend de N_r) suffit à provoquer

des écart de l'ordre de 10%.

La conformité entre les valeurs obtenues par simulation et les valeurs stationnaires théoriques permet de valider l'implémentation des modèles de simulation dans le simulateur DHS.

7.2.1.5 Influence du temps de réflexion

Nous nous intéressons maintenant à l'influence du temps de réflexion des clients, sur le fonctionnement du serveur. La figure 7.6 présente les comportements du serveur pour trois valeurs différentes de temps de réflexion : aucun temps de réflexion, un temps de réflexion de 68 secondes en moyenne et un temps de réflexion de 200 secondes en moyenne.



(a) Temps de réponse de la machine

(b) Nombre de requêtes dans le système

Fig. 7.6 – Influence du temps de réflexion

On peut s'apercevoir que la valeur de ce paramètre n'a aucune influence sur le comportement du système en régime permanent. Par contre, ce dernier influe sur le régime transitoire. Plus le temps de réflexion est important, plus le régime transitoire est long.

7.2.2 Serveur unique, clients différenciés

Nous nous intéressons cette fois ci à l'étude de la même machine hébergeant un serveur Web devant traiter des requêtes provenant de deux populations de clients ayant des comportements différents.

7.2.2.1 Modèle analytique

Le modèle analytique utilisé est le même que précédemment, dans lequel des classes d'application sont différenciées.

Le temps de réponse du serveur Web à une requête d'un client de la classe k est exprimé par

$$R^k_{serveur} = R^k_{proc} + R^k_{disque} + R^k_{E/S}. \tag{7.12}$$

Le temps de traitement d'une requête de classe k dans chacune des files processeur, disque et entrées/sorties est donnée par :

$$R^k_{proc} = \frac{Q^k_{proc}}{\lambda^k_r}, \quad R^k_{disque} = \frac{Q^k_{disque}}{\lambda^k_r}, \quad R^k_{E/S} = \frac{Q^k_{E/S}}{\lambda^k_r} \tag{7.13}$$

avec :

$$Q^k_{proc} = \frac{U^k_{proc}}{U_{proc}} \times Q_{proc}, \quad Q^k_{disque} = \frac{U^k_{disque}}{U_{disque}} \times Q_{disque}, \quad Q^k_{E/S} = \frac{U^k_{E/S}}{U_{E/S}} \times Q_{E/S} \tag{7.14}$$

$$Q_{proc} = \frac{U_{proc}}{1 - U_{proc}}, \quad Q_{disque} = \frac{U_{disque}}{1 - U_{disque}}, \quad Q_{E/S} = \frac{U_{E/S}}{1 - U_{E/S}} \tag{7.15}$$

$$U_{proc} = \Sigma_{k \in K} U^k_{proc}, \quad U_{disque} = \Sigma_{k \in K} U^k_{disque}, \quad U_{E/S} = \Sigma_{k \in K} U^k_{E/S} \tag{7.16}$$

$$U^k_{proc} = \frac{\lambda^k_r}{\mu^k_{proc}}, \quad U^k_{disque} = \frac{\lambda^k_r}{\mu^k_{disque}}, \quad U^k_{E/S} = \frac{\lambda^k_r}{\mu^k_{E/S}} \tag{7.17}$$

Le taux d'arrivée de requêtes de chaque classe sur le serveur noté λ^k_r est obtenu en faisant le produit du taux d'arrivée de sessions et du nombre moyen de requêtes par session :

$$\lambda^k_r = \Lambda^k_{session} \times N^k_r \tag{7.18}$$

7.2.2.2 Caractéristiques de la simulation

La première population de clients considérée (notée A) est celle traitée dans l'étude précédente (tableau 7.1). Les caractéristiques de la deuxième population de clients (notée B) sont résumées dans le tableau 7.2. Ces caractéristiques ont été choisies de manière arbitraire afin de représenter des clients accédant à des contenus de taille plus importante, de manière moins régulière que pour les clients A.

Paramètre	distribution	caractéristiques
Temps processeur	déterministe	0.5 s
Taille image mémoire	déterministe	5 Mo
Taille d'une requête	lognormale	moyenne 360 octets, variance 106,5.
Taille des pages	lognormale	moyenne 50 Ko
Arrivées de sessions	exponentielle	
Nombre de requêtes par session	exponentielle arrondie entre 1 et 200	moyenne 2
Période de réflexion	exponentielle	moyenne 120 s

Tab. 7.2 – Les différentes distributions utilisées pour la population de clients B

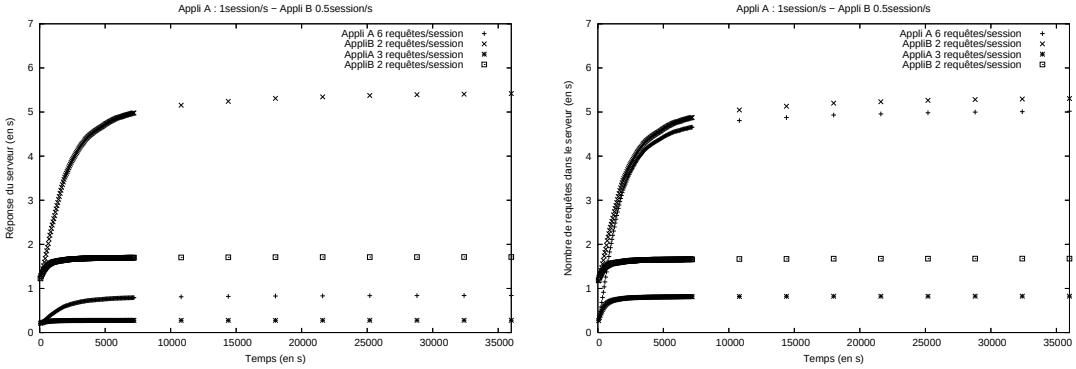
Nous présentons trois séries de test, permettant de faire varier la charge appliquée au serveur, et la part de cette charge imputée à chaque population de clients :

– Test 1 : taux d'arrivée de 1 session/seconde pour A, 0.5 session/seconde pour B. 1 requête par session pour A et B.

– Test 2 : taux d'arrivée de 1 session/seconde pour A, 0.5 session/seconde pour B. 3 requêtes par session pour A et 2 requêtes par sessions pour B.
– Test 3 : taux d'arrivée de 1 session/seconde pour A, 0.5 session/seconde pour B. 6 requêtes par session pour A et 2 requêtes par session pour B.

7.2.2.3 Comportement du serveur

La figure 7.7 présente les temps de réponses et les nombre de requêtes présentes dans le serveur pour chaque population de clients, pour les tests 2 et 3.



(a) Temps de réponse de la machine (b) Nombre de requêtes dans le système

Fig. 7.7 – Charge du système avec différentes populations de clients

La figure 7.8 représente le pourcentage d'écart entre les valeurs obtenues en régime stationnaire par simulation, à celle calculées analytiquement, pour les trois types d'expérimentation présentées précédemment. Comme précédemment, les marges d'erreurs sur les temps de réponse de l'application restent raisonnables ($< 10\%$).

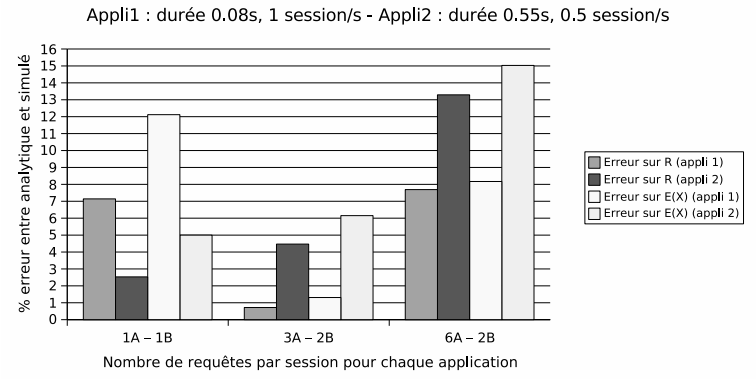


Fig. 7.8 – Comparaison entre résultats simulés et résultats analytiques

7.2.3 Partage de charge sur plusieurs machines

L'objectif de cette étude est d'étudier l'effet, sur le temps de réponse à une requête, de l'ajout de plusieurs serveurs se partageant les requêtes.

Les requêtes des clients sont adressées à un nœud frontal qui aiguille ces requêtes vers le serveur de son choix en utilisant soit un algorithme de partage de charge, soit un algorithme d'équilibrage de charge.

7.2.3.1 Caractéristiques de la simulation

Nous reprenons la simulation de la population de clients de type A traitée dans la section 7.2.1. Nous considérons le cas où le taux d'arrivée de session vaut 3 sessions/secondes, qui conduisait à la saturation d'une machine unique.

7.2.3.2 Résultats obtenus

Trois expérimentations sont menées :

1. Le comportement du système est analysé lorsque le nombre de machines augmente, en utilisant un algorithme de partage de charge. Toutes les machines ont les mêmes caractéristiques (figures 7.9(a), 7.9(b)).
2. Le comportement du système est analysé lorsque le nombre de machines augmente, en utilisant un algorithme de partage de charge. La grappe de machines simulée est divisée en deux groupes. La moitié des machines possèdent les mêmes caractéristiques que dans l'expérimentation précédente, et l'autre moitié est 1.5 fois plus rapide (en processeur et disque) (figures 7.9(c), 7.9(d)).
3. Le même test avec des machines hétérogènes est réalisé, mais cette fois ci avec un algorithme d'équilibrage de charge (figures 7.9(c), 7.9(d)).

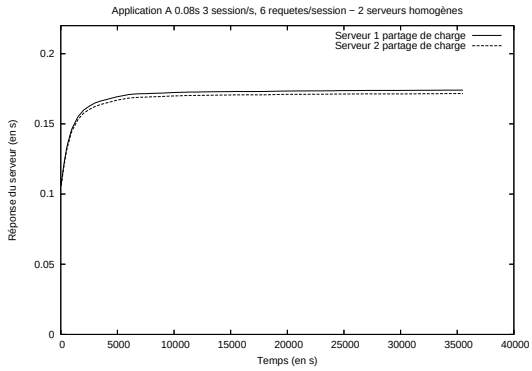
La première observation est que l'utilisation de plusieurs machines permet bien de traiter la charge de travail qui provoquait la saturation d'une machine unique. De plus, l'utilisation d'un nombre de machines supérieur permet de réduire le temps de réponse aux requêtes des clients.

L'utilisation de machines hétérogènes, et notamment de machines plus rapides a un effet différent selon l'algorithme d'ordonnancement utilisé.

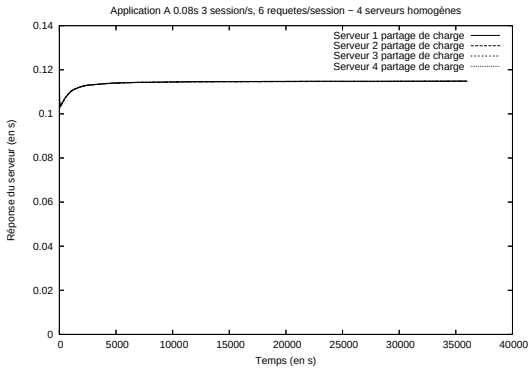
Lorsque l'on considère l'algorithme de partage de charge, L'utilisation de machines "rapides" permet d'observer un gain de performances pour les requêtes traitées par ces machines uniquement. Cependant, le temps de réponse des requêtes traitées par les machines "lentes" reste inchangé.

L'algorithme d'équilibrage de charge, au contraire, permet un gain de performances sur l'ensemble des requêtes. Ceci s'explique par le fait qu'une charge de travail supérieure est demandée aux machines "rapides", et de ce fait une charge de travail inférieure est demandée aux machines "lentes".

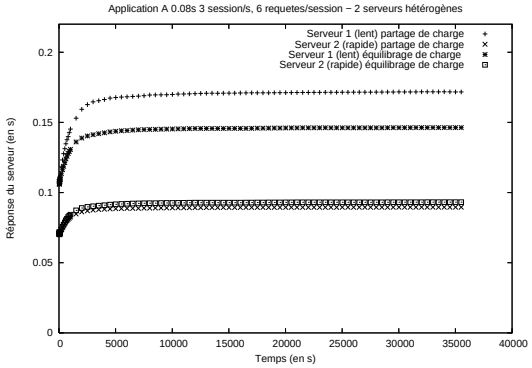
L'algorithme d'équilibrage de charge utilisé pour la simulation consiste à aiguiller la requête vers le serveur traitant le moins de clients. Ce n'est donc pas un véritable équilibrage de charge, pour lequel l'utilisation de chaque ressource du serveur devrait être prise en compte. De ce fait,



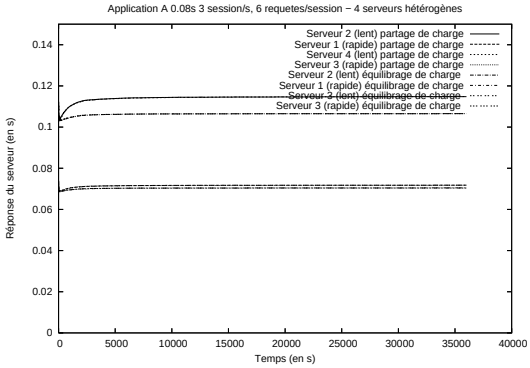
(a) 2 machines homogènes, partage de charge



(b) 4 machines homogènes, partage de charge



(c) 2 machines hétérogènes



(d) 4 machines hétérogènes

Fig. 7.9 – Réponse du système avec plusieurs machines

les temps de réponse sur chaque serveur, avec cet algorithme de placement ne sont pas tout à fait égaux, mais restent inférieurs à ceux obtenus avec l'algorithme de partage de charge.

7.2.3.3 Comparaison avec un modèle analytique

L'algorithme de partage de charge possède la particularité de répartir le nombre de requêtes de manière équitable entre chaque serveur de la grappe, indépendamment de la capacité du serveur en question.

De ce fait, il est possible de prédire le temps de réponse de chaque serveur en utilisant le modèle analytique présenté en section 7.2.1, en modifiant le calcul du taux d'arrivée de requêtes λ_r de la manière suivante :

$$\lambda_r = \frac{\Lambda_{session} \times N_r}{N_{serveurs}} \tag{7.19}$$

où $N_{serveurs}$ représente le nombre de machines de la grappe.

La figure 7.10 présente les pourcentages d'erreur entre les valeurs de régime stationnaire obtenues par simulation et par calcul analytique. Les valeurs présentées concernent uniquement l'algorithme de partage de charge, pour des machines homogènes ou hétérogènes.

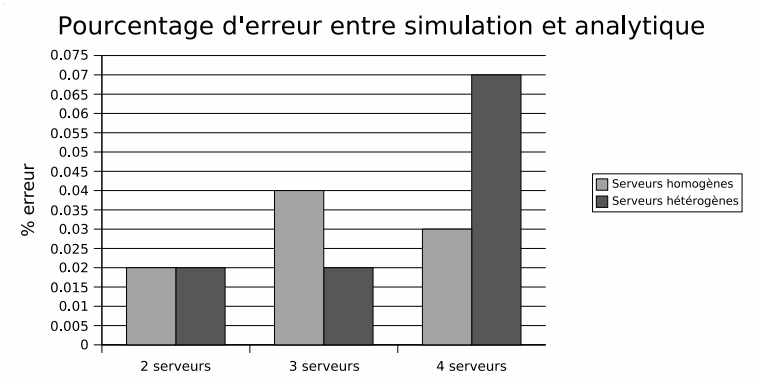


Fig. 7.10 – Comparaison entre résultats simulés et résultats analytiques

Une fois de plus, le comportement de la simulation est conforme aux résultats analytiques. Les modèles de simulation conçus permettent de prédire les performances d'applications séquentielles réparties sur un ensemble de machines pouvant avoir des performances hétérogènes, en tenant compte d'un algorithme d'ordonnancement pré-défini.

7.3 Applications parallèles

Les tests présentés dans cette section ont pour objectif de valider le modèle d'applications parallèles. Une attention particulière est portée à la validité de la simulation des communications réseau, et des communications bloquantes de la norme MPI.

Cette validation repose sur la comparaison entre les durées d'exécution réelle d'une application parallèle, sur une grappe de machines de référence, et l'estimation de ces durées obtenue par simulation. L'application parallèle utilisée est une application de test permettant de faire varier le grain de calcul.

7.3.1 Caractéristiques matérielles et logicielles de la grappe utilisée

L'architecture matérielle utilisée pour valider le test est composée d'une grappe regroupant huit machines homogènes. Un réseau Ethernet 1Gb/s inter-connecte les 8 machines de la grappe. Les caractéristiques des machines sont résumées dans le tableau 7.3,

Processeur	i686, 1,5 GHz
Mémoire vive	1 Go
Système d'exploitation	Linux Fedora Core 2, noyau 2.6.9

Tab. 7.3 – Architecture type d'un nœud de la grappe

Le système d'exploitation utilisé implémente les mécanismes de NAPI permettant de limiter le nombre d'interruptions générées par la carte réseau.

Les cartes réseau utilisées par les machines de la grappe sont des cartes réseau de la marque "Broadcom" utilisant le principe d'*interrupt mitigation*. Lorsque le noyau autorise le déclenchement d'interruptions (interruptions activées ou non par NAPI), ces cartes attendent la réception de 6 segments de données, ou l'expiration d'une alarme de $18\mu s$ depuis la réception du dernier segment de données avant de générer une interruption.

Le protocole TCP est paramétré avec les options par défaut de LAM/MPI (algorithme de Nagle désactivé). Les tailles des zones tampon d'émission et de réception de TCP sont positionnées au maximum entre la valeur définit par le système d'exploitation (16 Ko en émission et 87380 octets en réception, par défaut) et la somme entre la taille maximale d'un message court de LAM/MPI (64Ko, par défaut) et la taille d'une enveloppe de message de LAM/MPI (24 octets, par défaut). Les valeurs utilisées par défaut sont donc 65560 octets en émission et 87380 en réception.

7.3.2 L'application de test

L'application utilisée pour ces tests est un benchmark permettant de faire varier le grain de calcul. C'est une application maître/esclaves dont le comportement est décrit par l'algorithme 7.1.

Alg 7.1 Application maître/esclaves de test

Maître :

Tant que Le nombre d'itérations maximal n'est pas atteint **Faire**
    Pour Chaque esclave **Faire**
        Récupération d'un message
    Fin Pour
    Boucle de calcul
Fin Tant que
Envoi du message de résultat au client

Esclave :

Tant que Le nombre d'itérations maximal n'est pas atteint **Faire**
    - envoi d'un message au maître
    - boucle de calcul
Fin Tant que

La taille des messages échangés, le nombre d'itérations et la durée des boucles de calcul sont paramétrables afin de modifier le grain de l'application. La durée de la boucle de calcul de chaque tâche esclave diminue avec le nombre d'esclaves, de sorte que la charge de travail globale de l'application parallèle reste constante.

Le mode de passage de messages utilisé pour les communications entre les différentes tâches est le mode bloquant (primitives `MPI_Send` et `MPI_Recv` de la norme MPI). Les phases de calcul sont simulées par des boucles d'attente active.

7.3.3 Protocole de test

L'objectif est d'utiliser l'application de test afin d'étudier la capacité de passage à l'échelle de la grappe de machines considérée. Pour cela, l'application est exécutée sur un nombre de machines croissant. L'objectif est de déterminer le nombre de machines à partir duquel les performances du systèmes se dégradent.

7.3.3.1 Comportement de l'application

Il est important d'étudier les différents paramètres ayant une influence sur l'application étudiée, afin de choisir des tests permettant d'observer l'influence de ces différents paramètres.

L'application considérée peut se comporter de deux manières différentes :

- soit la tâche maître termine sa phase de calcul avant la réception des messages des esclaves, auquel cas seul le temps de calcul des esclaves est un facteur limitant de la performance,
- soit la durée de calcul de la tâche maître est supérieure à celle des tâches esclaves, auquel cas la tâche maître devient le facteur limitant de la performance. La tâche maître ne pouvant pas lire les messages suffisamment rapidement, son tampon de réception se remplit, ce qui a pour effet de bloquer l'émission des messages des esclaves et donc de bloquer leur traitement.

Selon le grain de calcul utilisé, les performances du réseau d'inter-connexion peuvent également devenir un facteur limitant de la performance. Outre le débit, ces performances dépendent des délais induits par les liens de transmission, des performances du switch de la grappe, et des performances de la pile TCP de chaque machine. Ces différents paramètres doivent être pris en compte si l'on souhaite pouvoir prédire les performances d'applications ayant une faible granularité.

Pour l'ensemble des tests présentés dans ce document, la taille des messages échangés est de 10 Ko. Cette valeur est suffisamment grande pour que chaque message soit segmenté en plusieurs segments de données (segments de taille 1460 octets par défaut).

7.3.3.2 Paramétrage du modèle réseau

Le délai dû aux liens et au switch de la grappe est estimé en réalisant un ping entre différentes machines de la grappe. Ce ping permet d'obtenir le *Round Trip Time* réseau. Ce délai est estimé à $\frac{RTT}{2}$. Lors de la simulation, il est équitablement réparti sur les deux liens présents de part et d'autre du switch.

Un test "à vide" est réalisé afin de déterminer le temps processeur consommé par la pile TCP. Ce test consiste à exécuter l'application avec des temps de calcul nuls pour le maître et les esclaves. Une étude type "boite noire" de l'application est réalisée afin de déterminer le temps processeur consommé par l'envoi et la réception des messages.

7.3.3.3 Esclaves lents

Cette première série de tests vise à étudier les performances du système lorsque le client est le facteur limitant des performances de l'application.

Les temps de calcul du maître et des esclaves sont estimés à partir du temps processeur consommé par chaque tâche. Ces valeurs sont mesurées une seule fois, pour paramétrer la simulation en considérant une seule tâche esclave, et un nombre d'itérations de calcul faible, de sorte que le temps mesuré ne soit pas influencé par le temps consommé par les communications réseau.

Le même test est réalisé avec trois valeurs différentes pour le nombre d'itérations, afin de modifier le grain de l'application. Les caractéristiques de ces trois tests sont détaillées dans le tableau 7.4.

Numéro du test	Taille des messages	Nombre d'itérations	Temps de calcul maître	Temps de calcul esclave	Temps calcul/ Temps communication
1	10Ko	200	66	265	$4000 \leqslant t \leqslant 17000$
2	10Ko	20000	66	265	$170 \leqslant t \leqslant 43$
3	10Ko	200000	66	265	$17 \leqslant t \leqslant 4$

Tab. 7.4 – Paramètres du test esclaves lents

7.3.3.4 Maître lent

Cette deuxième série de tests vise à étudier les performances du système lorsque le maître est le facteur limitant des performances de l'application.

Les paramètres du modèle de simulation sont obtenus de la même manière que pour la série de tests précédente. Ces paramètres sont détaillés dans le tableau 7.5.

Numéro du test	Taille des messages	Nombre d'itérations	Temps de calcul maître	Temps de calcul esclave	Temps calcul/ Temps communication
4	10Ko	200	265	66	≈ 17000
5	10Ko	20000	66	265	≈ 174
6	10Ko	200000	66	265	≈ 17

Tab. 7.5 – Paramètres du test maître lent

7.3.4 Résultats obtenus

7.3.4.1 Application gros grain

Les figures 7.11 et 7.12 présentent les résultats obtenus pour une application gros grain (tests 1 et 4).

Les valeurs simulées sont obtenues en considérant le débit des interfaces réseau égal à 1Gb/s (débit réel) et en simulant un temps d'accès au processeur lors de chaque lecture ou écriture d'un message égal à 0.1 ms. Cette valeur est obtenue en étudiant le temps processeur consommé par le système à vide (sans temps de calcul) et en le divisant par le nombre d'itérations de calcul.

On observe, pour le test numéro 1 (figure 7.11), un gain de calcul jusqu'à 5 tâches. Par la suite, le temps de calcul de chaque tâche esclave devient inférieur au temps de calcul de la tâche maître. L'application est alors limitée par le temps de calcul du maître (66 s).

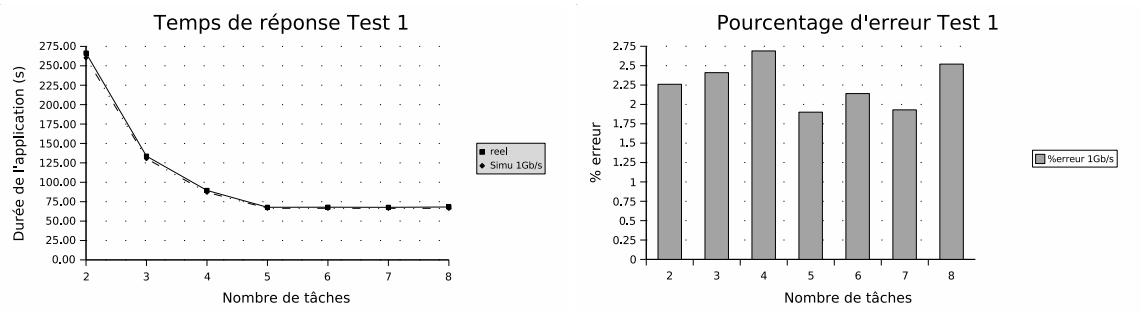


Fig. 7.11 – Application gros grain, esclaves lents

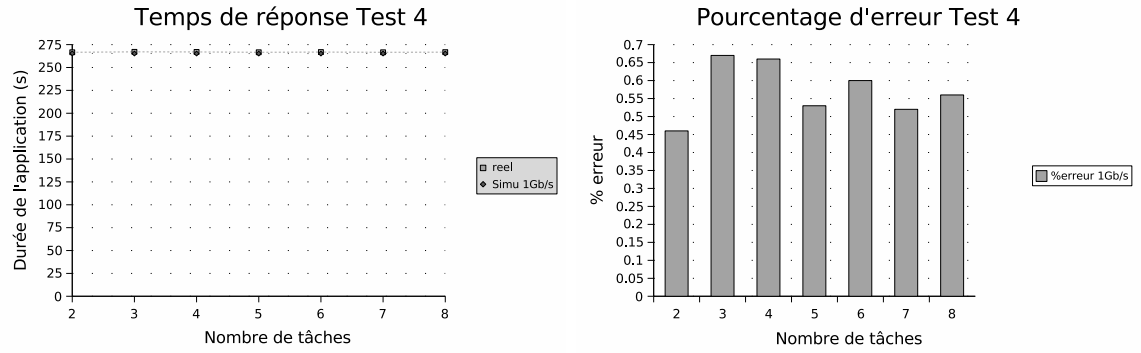


Fig. 7.12 – Application gros grain, maître lent

Si l'on considère le test numéro 4 (figure 7.12), l'application est limitée dès le départ par la tâche maître, de ce fait aucun gain n'est observable lorsque l'on accroît le nombre de tâches. Le nombre d'itérations de calcul étant relativement faible, le temps de traitement de l'application reste quasiment constant quelque soit le nombre de tâches considéré.

Dans les deux cas de figure considérés, les résultats obtenus par simulation sont très proche de la réalité (moins de 3% d'erreur).

7.3.4.2 Application grain moyen

Les figures 7.13 et 7.14 présentent les résultats obtenus pour une application moyen grain (tests 2 et 5).

Trois résultats de simulations sont considérés :

- le premier intitulé "simu 1Gb/s" considère des interfaces réseau ayant un débit de 1Gb/s et un temps de lecture et d'écriture de chaque message égal à 0.1 ms,
- le deuxième intitulé "simu 400Mb/s" considère cette fois ci des interfaces réseau ayant un débit de 400Mb/s. Ce débit correspond à la bande passante réelle mesurée en utilisant l'application à vide (sans temps de calcul). Le temps de lecture et d'écriture de chaque message est toujours égal à 0.1 ms,
- le troisième intitulé "simu TCP/stack" considère des interfaces réseau ayant un débit de 1Gb/s, et simule le comportement de la pile TCP. Le comportement des *softIRQ* d'émission et de réception de la pile TCP est simulé. L'émission d'un segment TCP est réalisé après un temps de traitement réalisé par la *softIRQ* d'émission. De la même

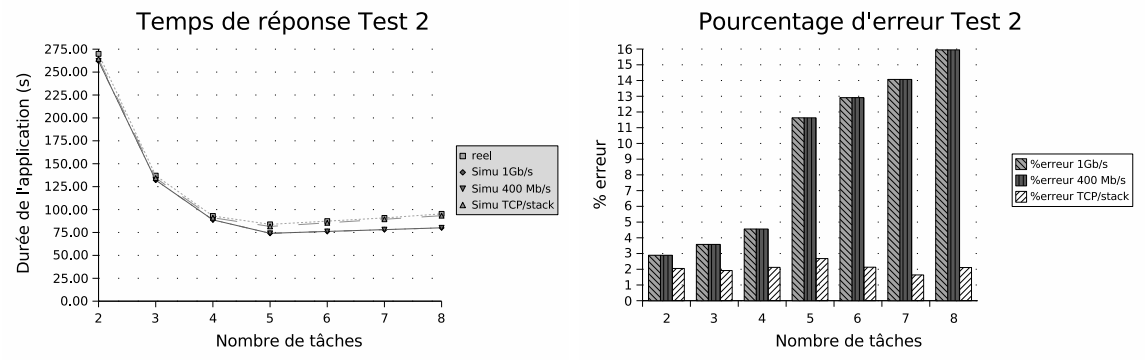


Fig. 7.13 – Application moyen grain, esclaves lents

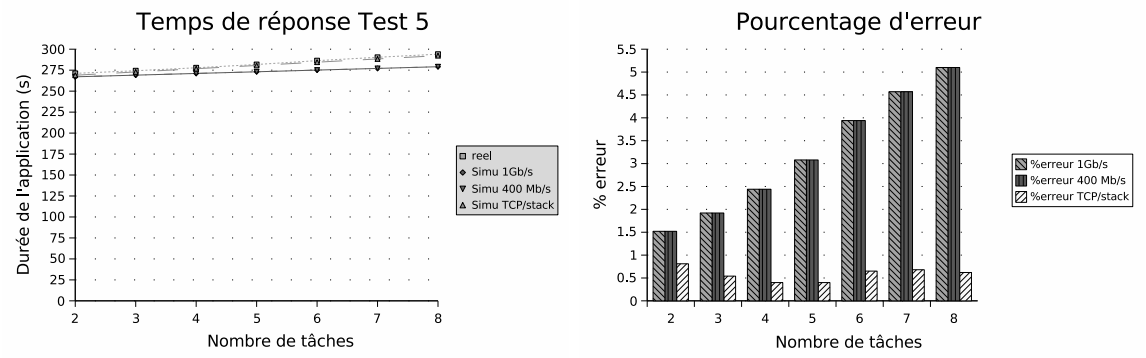


Fig. 7.14 – Application moyen grain, maître lent

manière, lors de la réception d'un segment TCP ce dernier n'est accessible dans le tampon de réception de l'application qu'après l'exécution d'un temps de traitement par la *softIRQ* de réception. Le mécanisme de NAPI est également simulé : lorsque la *softIRQ* de réception accède au processeur, elle lit toutes les données présentes sur la carte réseau jusqu'à l'expiration de son quantum. Les temps de traitement associés aux *softIRQs* est une valeur proportionnelle à la taille du segment traité. La valeur de ce délai est estimé en analysant le comportement du système à vide (sans temps de calcul), et est fixée à 62.5 Mo/s. Un temps de lecture et d'écriture de chaque message est également fixé à 0.03 ms.

On observe, pour le test numéro 2 (figure 7.13), un gain de calcul jusqu'à 5 tâches. Par la suite, le temps de calcul de chaque tâche esclave devient inférieur au temps de calcul de la tâche maître. L'application est alors limitée par le temps de calcul du maître (66 s). Le temps de traitement de la tâche maître correspond à son temps de calcul, auquel s'additionne le temps de traitement imposé par le pile TCP lors de la réception de nouveaux segments de données. De ce fait, le temps minimum de traitement de l'application n'est plus de 66 s comme pour le cas gros grain, mais 83 s. Enfin, lorsque l'on augmente le nombre de tâches, le nombre de segments TCP traités par la tâche maître augmente et par conséquent la durée totale de l'application également.

Le même comportement est observé pour le test numéro 5 (figure 7.14), le temps de traitement de l'application augmente avec le nombre de tâches esclaves, du fait du temps processeur consommé par la pile TCP.

La qualité de la prédiction obtenue par simulation diffère en fonction du modèle de simulation utilisé. Lorsque l'on ne tient pas compte de l'influence de la pile TCP, plus le nombre de tâches augmente, plus l'écart entre les valeurs simulées et mesurées augmente jusqu'à atteindre 16% pour 8 tâches. Cependant, la limite à partir de laquelle les performances de l'application commencent à se détériorer reste la même, en simulation et en réel.

Les résultats obtenus en simulant le comportement de la pile TCP sont par contre bien meilleurs (inférieurs à 2% d'erreur).

7.3.4.3 Application grain fin

Les figures 7.15 et 7.16 présentent les résultats obtenus pour une application grain fin (tests 3 et 6).

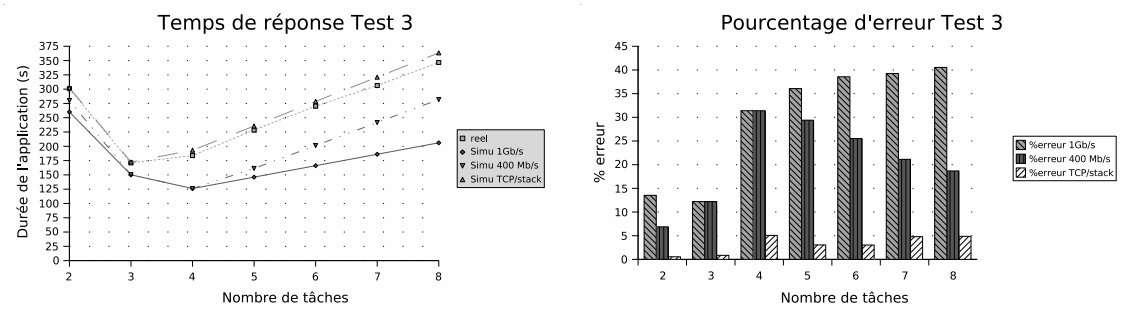


Fig. 7.15 – Application grain fin, esclaves lents

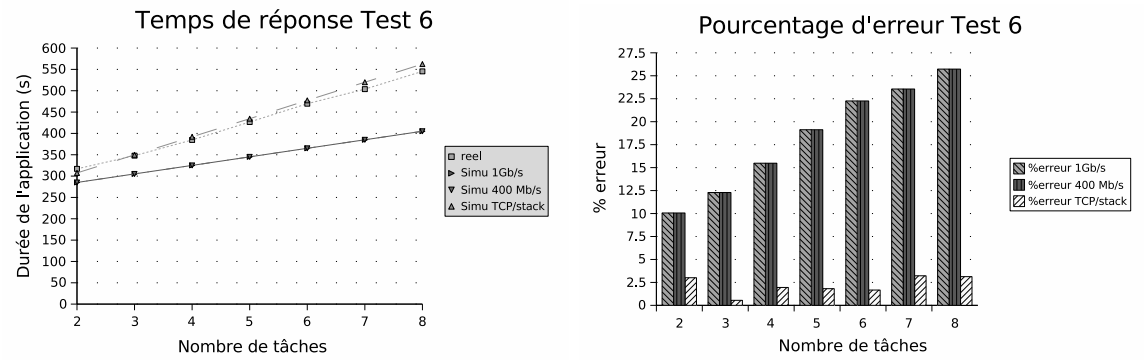


Fig. 7.16 – Application grain fin, maître lent

Trois résultats de simulations sont considérés, comme pour le grain moyen : "simu 1Gb/s", "simu 400Mb/s", "simu TCP/stack".

On observe cette fois ci, pour le test numéro 3 (figure 7.15), un gain de temps jusqu'à seulement trois tâches. Le temps de calcul le plus faible obtenu est alors égal à 170 s, ce

qui correspond au temps de calcul des esclaves (132 s) auquel s'ajoute un temps imputé aux communications. Le goulot d'étranglement n'est plus l'application, mais le réseau.

Pour le test numéro 6 (figure 7.16), comme précédemment, le temps de traitement de l'application croit en fonction du nombre de tâches considéré, du fait du temps processeur consommé par la pile TCP. Cependant, le nombre de messages échangés étant supérieur au cas moyen grain, l'impact de la pile TCP est supérieur.

La qualité de la prédiction obtenue par simulation diffère en fonction du modèle de simulation utilisé. Lorsque l'on ne tient pas compte de l'influence de la pile TCP, plus le nombre de tâches augmente, plus l'écart entre les valeurs simulées et mesurées augmente jusqu'à atteindre 40% pour 8 tâches. Cet écart peut être minimisé en simulant une bande passante de 400 Mb/s, mais il reste tout de même conséquent (30%). Dans les deux cas, la simulation ne permet pas de déterminer la limite à partir de laquelle les performances de l'application commencent à se détériorer.

Les résultats obtenus en simulant le comportement de la pile TCP sont par contre bien meilleurs (inférieurs à 5% d'erreur). Dans ce cas, la limite de perte de performances est correctement détectée, ce qui démontre l'importance de la pile TCP lorsque l'on s'intéresse à des applications de grain fin.

7.3.4.4 Performances du simulateur

Nous présentons ici, une comparaison entre le temps d'obtention d'une valeur simulée et le temps d'obtention de cette même valeur en utilisant le système réel. Le but de cette comparaison est d'évaluer l'utilisabilité du simulateur. Les résultats (figure 7.17) sont présentés à la fois pour des applications effectuant des calculs, et pour des applications pour lesquelles seules les communications réseau interviennent.

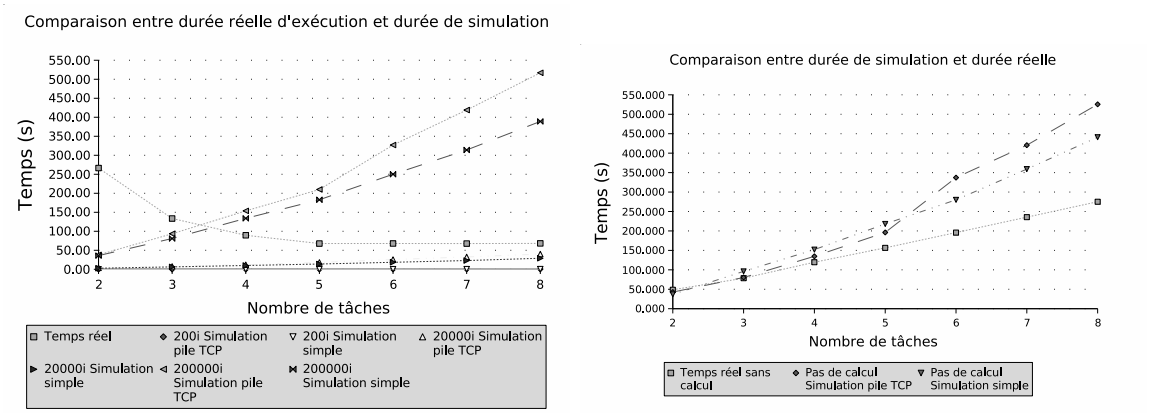


Fig. 7.17 – Durée de la simulation

Une première constatation est que, si pour le système réel le temps de traitement peut décroître lorsque le nombre de tâches augmente, le temps de simulation lui augmente toujours lorsque le nombre de tâches augmente (le nombre d'événements à traiter est plus important).

Les performances du simulateur DHS dépendent essentiellement du nombre d'événements traités par le simulateur (un événement peut être l'arrivée d'un paquet dans une file, sa sortie de la file, l'expiration d'une alarme ...). La partie du simulateur la plus consommatrice en

événements est la simulation du protocole TCP pour lequel plusieurs événements sont générés pour chaque segment de données transféré (événements gérant le cheminement du segment dans les différentes files, événements pour les alarmes TCP ...). De ce fait, les performances du simulateur sont largement impactées par la taille et le nombre des messages échangés. Le tableau 7.6 donne par exemple un aperçu du nombre d'événements DHS utilisés par une même application en fonction du nombre d'itérations de calcul réalisé et le temps de simulation associé.

Nombre d'itérations	Nombre de tâches	Nombre d'événements	Temps de simulation (s)
200	2	17736	0.079
200	8	132655	0.436
20000	2	1875803	3.126
20000	8	14929071	36.5
200000	2	20235974	37
200000	8	151369197	441

Tab. 7.6 – Lien entre communications, événements DHS et temps de simulation

Pour cette raison, comme l'illustre la figure 7.17, le gain de temps offert par la simulation, par rapport à une exécution réelle varie énormément selon la masse de données échangée par l'application. Lorsque l'on ne considère que les communications (pas de calcul), la durée de simulation reste sensiblement égale à la durée de l'application réelle, jusqu'à 4 machines. Par la suite, la durée de simulation devient supérieure à la durée réelle d'exécution jusqu'à atteindre le double pour 8 machines. Par contre, lorsque l'on considère des applications avec un grain de calcul moyen ou gros, le gain est conséquent et le temps de simulation reste faible (40 s maximum).

7.4 Conclusion

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

La deuxième étude a permis de valider le modèle d'applications parallèles, et notamment la prise en compte des communications réseaux entres les applications. Cette étude a permis de mettre en évidence la qualité des prédictions obtenues en fonction du niveau de détails du modèle de simulation utilisé et du grain de l'application.

Enfin, une étude des performances du simulateur a été menée, afin d'évaluer le rapport entre précision des résultats obtenus et temps d'obtention de ces résultats. Le code du simulateur DHS utilisé pour cette étude de performances est un code ayant subit certaines optimisations, mais pouvant être certainement optimisé afin d'obtenir certains gains de performances.

Chapitre 8

Conclusion.

8.1 Gestion de ressources en mode ASP

Le gestionnaire de ressources Aroma utilisable en mode ASP a été développé dans le cadre de cette thèse. Sa structure hiérarchique, associée à l'utilisation du protocole de communication JINI, lui permettent de s'adapter facilement aux différents changements de structure des grappes de calculs ; qu'il s'agisse de changements dûs à des interventions humaines (ajout, suppression de nœuds de calculs), ou à des pannes du système (panne matérielle, logicielle, réseau). Dans le cas de pannes, le système de réplicas mis en place permet de maintenir un état cohérent des grappes de machines et de limiter au maximum les informations perdues. De plus, sa conception sous forme de *plugins* facilite l'ajout ou la modification à chaud de nouvelles fonctionnalités.

Aroma offre de nombreux services permettant de connaître précisément l'état d'une grappe : observation détaillée de l'état actuel et passé de chaque machine, suivi de l'état d'exécution des différentes applications lancées par le biais du gestionnaire. Ces nombreuses informations permettent l'utilisation d'algorithmes de placement de tâches appréhendant la notion de qualité de service.

Enfin, les mécanismes d'authentification, de gestion des droits et des ressources des utilisateurs des grappes de machines permettent l'utilisation de ce gestionnaire de ressource en mode ASP.

8.2 Évaluation de performances d'applications

Cette thèse aborde la problématique du dimensionnement de grappes de machines. Cette problématique est traitée par le biais de la simulation. Divers modèles du comportement des clients, du comportement des applications, du fonctionnement des machines, du système d'exploitation et des protocoles de communication utilisés sont proposés.

La qualité des prédictions obtenues par simulation ainsi que les performances du simulateur sont étudiées. L'originalité de ce travail réside dans le fait de pouvoir simuler le comportement d'une grappe de machine dans sa totalité. Il est ainsi possible de simuler le fonctionnement d'un système ASP répondant à des requêtes provenant de clients ayant des comportements

totalement différents, exécutant des applications aux comportements tout aussi différents, sur des machines pouvant être hétérogènes. Ce simulateur peut également être utilisé afin de comparer les performances de divers algorithmes de placement pour grappes.

8.3 Perspectives

Le niveau de tolérance au faute du gestionnaire de ressources Aroma pourrait être augmenté, notamment en permettant la récupération d'applications s'exécutant sur des machines tombées en panne. Ceci nécessiterait la mise en place de mécanismes de *checkpointing*, permettant la reprise de l'exécution sur une machine différente.

Les performances du simulateur DHS peuvent être améliorées, soit en essayant d'optimiser un peu plus le code du simulateur, soit en parallélisant le noyau de calcul afin de tirer profit de la puissance offerte par les grappes de machines.

Des modèles analytiques des applications parallèles doivent être développés. Ces modèles permettront de tenir compte de la prédiction du temps d'exécution d'une application lors du placement de cette dernière. En effet, les algorithmes de placement traitant de la notion de qualité de services, implémentés dans Aroma, ont besoin d'avoir une estimation du temps de traitement des tâches à placer afin de garantir les différents niveaux de qualité de services (dates butoirs ...). Ces estimations sont actuellement des valeurs fournies par le client du système ASP. Des modèles simplifiés du comportement des applications placées, permettant une estimation en temps réel de la durée d'exécution de l'application pourraient être utilisées afin de résoudre ce problème.

Annexe A

Fichiers de configuration des capteurs de charge

Ces fichiers de configuration permettent le chargement à chaud des capteurs de charge au sein des serveurs d'Aroma. Ils sont écrits en XML, et doivent respecter un schéma XSD que nous définirons. Il existe deux types de fichiers de configuration que nous décrirons successivement :

- la liste des capteurs de charge disponible,
- les éléments de configuration de chaque ressource.

A.1 Liste des capteurs de charge

Ce type de fichier contient la liste des ressources à observer, avec pour chacune d'elles les informations suivantes :

- le nom de la ressource,
- le nom du fichier JAR contenant le capteur,
- le nom du fichier de configuration du capteur.

Nous pouvons, d'après ces éléments, établir la structure des fichiers XML correspondants. Elle est donnée par le schéma XSD suivant :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="ResourcesList" type="ResourcesListType"/>

    <xsd:complexType name="ResourcesListType">
        <xsd:sequence>
            <xsd:element name="Resource" type="ResourceType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="ResourceType">
        <xsd:sequence>
```

```
            <xsd:element name="Name" type="xsd:string"/>
            <xsd:element name="JarFile" type="xsd:string"/>
            <xsd:element name="XmlFile" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

Afin de mieux illustrer ceci, voici un exemple de fichier XML contenant une liste de capteurs de charge :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<ResourcesList>

    <Resource>
        <Name>cpu</Name>
        <JarFile>cpuwatcher.jar</JarFile>
        <XmlFile>aroma/resourceunit/watcher/cpuwatcher.xml</XmlFile>
    </Resource>

    <Resource>
        <Name>memory</Name>
        <JarFile>memwatcher.jar</JarFile>
        <XmlFile>aroma/resourceunit/watcher/memwatcher.xml</XmlFile>
    </Resource>

    <Resource>
        <Name>processes</Name>
        <JarFile>processwatcher.jar</JarFile>
        <XmlFile>aroma/resourceunit/watcher/processwatcher.xml</XmlFile>
    </Resource>

</ResourcesList>
```

A.2 Configuration des capteurs de charge

Les fichiers de configuration des capteurs de charge contiennent diverses informations sur ces derniers. Il en existe un par capteur. Les informations que l'on peut y trouver sont les suivantes :

- le nom de classe implémentant le capteur,
- une description de la ressource observée,
- des informations sur l'accès aux différents champs de la ressource,

- des informations sur le stockage des différents champs dans la base de données,
- la légende des graphiques de l'observateur de ressources pour chacun des champs,
- des informations sur la construction de ces graphiques (échelle, titre des axes, unités, etc.).

Nous pouvons, d'après ces éléments, établir la structure des fichiers XML correspondants. Elle est donnée par le schéma XSD suivant :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="ResourceConfiguration" type="ResourceConfigurationType"/>

    <xsd:complexType name="ResourceConfigurationType">
        <xsd:sequence>
            <xsd:element name="ClassName" type="xsd:string"/>
            <xsd:element name="Description" type="xsd:string"/>
            <xsd:element name="Keys" type="KeysType"/>
            <xsd:element name="Captions" type="CaptionsType"/>
            <xsd:element name="DBColumns" type="DBColumnsType"/>
            <xsd:element name="GraphRecorder" type="GraphRecorderType"/>
            <xsd:element name="PieChart" type="PieChartType"/>
            <xsd:element name="Chart" type="ChartType"/>
            <xsd:element name="KiviatDiagram" type="KiviatDiagramType"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="KeysType">
        <xsd:sequence>
            <xsd:element name="Key" type="xsd:string" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="CaptionsType">
        <xsd:sequence>
            <xsd:element name="Caption" type="xsd:string" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="DBColumnsType">
        <xsd:sequence>
            <xsd:element name="DBColumn" type="xsd:string" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="GraphRecorderType">
        <xsd:sequence>
```

```
            <xsd:element name="inXScale" type="xsd:string"/>
            <xsd:element name="highFrequence" type="xsd:string"/>
            <xsd:element name="yUnity" type="xsd:string"/>
            <xsd:element name="unity" type="xsd:string"/>
            <xsd:element name="inCapacity" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="PieChartType">
        <xsd:sequence>
            <xsd:element name="inXScale" type="xsd:string"/>
            <xsd:element name="inUnity" type="xsd:string"/>
            <xsd:element name="inCapacity" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="ChartType">
        <xsd:sequence>
            <xsd:element name="inXScale" type="xsd:string"/>
            <xsd:element name="inUnity" type="xsd:string"/>
            <xsd:element name="inCapacity" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="KiviatDiagramType">
        <xsd:sequence>
            <xsd:element name="inCapacity" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

Afin de mieux illustrer ceci, voici un exemple de fichier XML contenant la configuration d'un capteur de charge (celui observant le taux d'utilisation du processeur) :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<ResourceConfiguration>

    <ClassName>aroma.resourceunit.watcher.CpuLoad</ClassName>

    <Description>Cpu load</Description>

    <Keys>
        <Key>nice</Key>
```

```
        <Key>user</Key>
        <Key>kernel</Key>
        <Key>idle</Key>
    </Keys>

    <Captions>
        <Caption>nice</Caption>
        <Caption>user</Caption>
        <Caption>kernel</Caption>
        <Caption>idle</Caption>
    </Captions>

    <DBColumns>
        <DBColumn>cpunice</DBColumn>
        <DBColumn>cpuusr</DBColumn>
        <DBColumn>cpusyst</DBColumn>
        <DBColumn>cpuidle</DBColumn>
    </DBColumns>

    <GraphRecorder>
        <inXScale>6</inXScale>
        <highFrequence>60</highFrequence>
        <yUnity>percents</yUnity>
        <unity>seconds</unity>
        <inCapacity>1000</inCapacity>
    </GraphRecorder>

    <PieChart>
        <inXScale>6</inXScale>
        <inUnity>seconds</inUnity>
        <inCapacity>100</inCapacity>
    </PieChart>

    <Chart>
        <inXScale>6</inXScale>
        <inUnity>percents</inUnity>
        <inCapacity>1</inCapacity>
    </Chart>

    <KiviatDiagram>
        <inCapacity>10</inCapacity>
    </KiviatDiagram>

</ResourceConfiguration>
```

Annexe B

Rappel des principes de base de TCP

Le but de cette annexe est de rappeler le fonctionnement des principaux mécanismes de TCP cités dans cette thèse. Des informations plus détaillées sont fournis par les RFCs [31][32][36][35][37].

B.1 Concepts de base

TCP fournit un canal de communication entre deux processus applicatifs. Ce canal est fiable, sans erreur et bidirectionnel avec contrôle et retransmission des données effectués aux extrémités de la liaison.

Plusieurs mécanismes permettent aux couches supérieures de ne pas se préoccuper du contrôle et de la validité des données reçues par la couche de niveau transport TCP. Les mécanismes peuvent se résumer ainsi :

- Les tailles des paquets assemblés par la couche TCP et transmis à la couche IP ne correspondent pas forcément aux tailles des paquets reçus des couches supérieures (utilisateur, application), et ceci à des fins d'optimisation de la bande passante. Un paquet transmis par la couche TCP à la couche IP est appelé un segment de donnée,
- A l'émission d'un segment, un temporisateur est armé. A l'expiration de celui-ci, et si le correspondant n'a pas renvoyé d'accusé de réception pour ce segment, il est considéré comme étant perdu, et est retransmis,
- Quand la couche TCP reçoit une donnée du réseau, un accusé-réception est envoyé après un délai optionnel. Ce délai autorise la prise en compte de nouvelles données à envoyer, afin d'optimiser la bande passante. Il ne doit pas excéder 500ms, et est habituellement de 200ms,
- Chaque segment est accompagné de son *checksum*. Un segment arrivant avec un *checksum* invalide est détruit et ignoré. Le temporisateur de l'émetteur détectera alors la perte,
- Le protocole IP ne garantie pas que l'ordre de réception des paquets à la destination est le même que l'ordre d'émission à la source. Les segments TCP peuvent donc arriver à la destination dans le désordre, et dans ce cas, la couche TCP se charge de les remettre dans l'ordre afin que l'application reçoive correctement les données,
- Les segments reçus en double sont éliminés,
- Chaque extrémité de la connexion TCP a un tampon de réception de taille limitée. TCP assure qu'une extrémité n'enverra pas plus de données que ce que son correspondant ne

peut recevoir.

B.2 Contrôle du débit d'émission

B.2.1 Objectif

L'objectif du protocole TCP est de maximiser l'efficacité du transfert de données. Ceci signifie que le protocole doit déterminer le point d'équilibre pour lequel le débit d'émission est maximisé, tout en conservant un nombre de pertes de paquets limité.

Augmenter le débit d'émission au delà de ce point d'équilibre risque de générer une congestion du réseau, ce qui a pour effet d'augmenter le nombre de pertes de paquets. Ceci forcera TCP à re-transmettre les données perdues, et donc à réduire l'efficacité du transfert de données.

A l'opposé, tenter d'éliminer complètement les pertes de paquets implique une réduction du taux d'émission. Ceci risque de sous-utiliser le réseau.

L'objectif de TCP est de synchroniser l'émetteur et le récepteur des données, de sorte que l'émetteur poste un paquet sur le réseau au même instant que le récepteur retire un paquet du réseau. Si l'émetteur émet plus rapidement, le phénomène de congestion apparait, s'il émet moins rapidement que le récepteur lit, les performances chutent.

Cette notion de synchronisation est implémentée par la notion de fenêtre glissante de TCP.

B.2.2 Fenêtre glissante

La synchronisation entre l'émétteur et le récepteur de données transportées par TCP est réalisée en utilisant les mécanismes d'accusé de réception (*acknowledgement*) et de fenêtre glissante (*sliding window*). Lorsqu'un segment de données est envoyé par l'émetteur, ce dernier est informé de la lecture de ce segment par le récepteur lors de la réception d'un packet d'accusé de réception correspondant. Attendre systématiquement la réception d'un accusé de réception avant d'émettre le segment de données suivant étant très pénalisant, le mécanisme de fenêtre glissante a été défini. Ce mécanisme permet d'envoyer un certain nombre de segments de données (appelé fenêtre glissante) sans attendre la réception d'accusés de réception.

Notons ack le numéro du dernier paquet dont on a reçu un accusé de réception, seq le numéro du prochain segment de données prêt à être envoyé et $window$ la taille de la fenêtre glissante. Alors, à tout instant, le numéro de séquence du dernier segment de données (noté seq_{last}) pouvant être envoyé est égal :

$$seq_{last} = ack + window - seq \tag{B.1}$$

La valeur de ack, et donc de seq_{last} augmente à chaque réception d'un accusé de réception, provoquant le décalage de la fenêtre glissante (d'où son nom). La figure B.1 illustre ce mécanisme.

Lorsque le transfert de données atteint le débit maximal autorisé par le réseau, la valeur de la fenêtre glissante représente le nombre maximal de segments de données pouvant être envoyés sans dégrader les performances du transfert.

En pratique, la valeur de la fenêtre glissante correspond au minimum entre la valeur de la fenêtre de réception (champ *rwnd* de l'entête d'un paquet TCP) et la valeur de la fenêtre de congestion (champ *cwnd* de l'entête d'un paquet TCP).

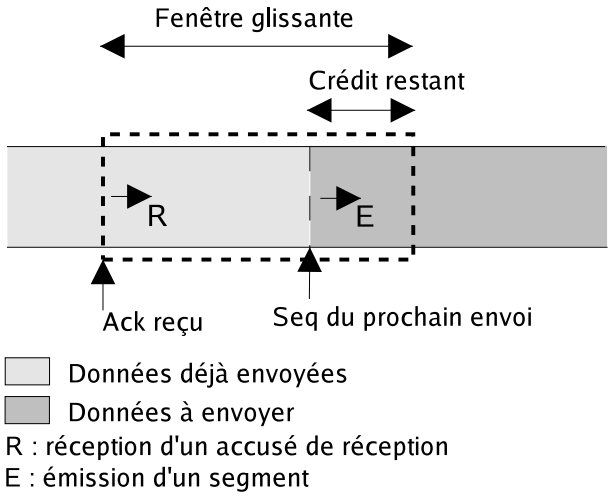


Fig. B.1 – Concept de fenêtre glissante

La fenêtre de réception correspond à l'espace mémoire libre dans la zone tampon de réception du destinataire des données. Cette valeur est mise à jour à chaque réception d'un accusé de réception.

La fenêtre de congestion peut être assimilée au débit sortant par RTT (*Round Trip Time*). Sa valeur évolue à chaque réception d'un accusé de réception. Deux mode coexistent dans TCP NewReno : *Slow Start* et *Congestion Avoidance*.

B.2.3 Fenêtre de congestion

Le mode *Slow-Start* permet de découvrir la bande passante disponible, en augmentant de manière agressive la valeur du débit *cwnd* (cette appellation provient du fait que la découverte se fait à partir d'un rythme très lent : 1 seul paquet pendant le premier RTT). Dans ce mode, *cwnd* est augmenté d'un paquet (mss) lors de la réception d'un accusé de réception (figure B.2).

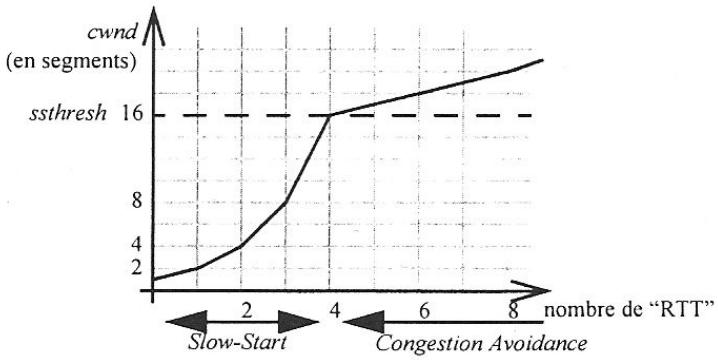


Fig. B.2 – Slow-start et Congestion-Avoidance

Lorsque le débit *cwnd* dépasse le seuil *ssthresh* (*Slow Start THRESHold*), on sort du mode *Slow-Start* pour entrer dans le mode *Congestion Avoidance*. Le principe dans ce mode est

de considérer qu'un régime stationnaire est atteint, en ne faisant pas augmenter le débit de manière trop forte, mais assez pour exploiter une éventuelle libération de la bande passante de bout-en-bout. Pour ce faire, *cwnd* est augmenté de 1/cwnd paquet (en réalité mss*mss/cwnd octets), ce qui offre un crédit d'un peu plus d'un paquet lors de la réception d'un accusé de réception.

B.3 Algorithme de Nagle

La taille des segments de données transmis par TCP peut être très variable, de 40 octets (taille de l'entête TCP/IP) à mss+40 (taille maximum). Dans le cas d'une grande distance entre les deux extrémités du transfert, un trop grand nombre de petits paquets est un gaspillage de la bande passante. L'algorithme de Nagle consiste à retarder l'émission des paquets de petite taille (typiquement de taille $mss/2$) jusqu'à ce qu'un paquet de taille plus conséquente puisse être envoyé, ou jusqu'à expiration d'une alarme (typiquement de 200 ms).

Bibliographie

[1] J. Aman, C. Eilert, D. Emmes, P. Yocom, and D. Dillenberg. Adaptative algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2) :242–283, 1997.

[2] Math B. An empirical model of http network traffic. In *IEEE INFOCOM 97*, volume 2, pages 592–600, 1997.

[3] David A. Bacigalupo, Stephen A. Jarvis, Ligang He, Daniel P. Spooner, Donna N. Dillenberger, and Graham R. Nudd. An investigation into the application of different performance prediction methods to distributed enterprise applications. *The Journal of Supercomputing*, 34(2) :93–111, 2005.

[4] A. Bayucan and R.L. Henderson. Portable batch system. Technical report, PBS, 1999.

[5] B.Miegemolle. Etude de modèles économiques pour les grilles de calcul. Master's thesis, INSA Toulouse, Septembre 2004.

[6] B.Miegemolle, T.Monteil, and S.Richard. Mise en place du mode failover pour le gestionnaire de ressources. Livrable Projet RNTL CASP No12, LAAS-CNRS, Septembre 2003.

[7] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Iona Michael Champion, Chris Ferris, and David Orchard. Web services architecture. Technical report, W3C Working Group, 2004.

[8] Greg Burns, Raja Daoud, and James Vaigl. LAM : An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[9] Site web du projet rntl casp, http ://www.laas.fr/casp/.

[10] Steve J. Chapin, Dimitrios Katramatos, John Karpovitch, and Andrew S. Grimshaw. The legion resource management system. In *Proceeedings Job Scheduling strategies for parallel processing workshop*, 1999.

[11] Arnold D., Agrawal S., Blackford S., Dongarra J., Miller M., Seymour K., Sagi K., Shi Z., and Vadhiyar S. Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[12] PJ. Denning. *Operating Systems Theory*. Prentice Hall, 1990.

[13] Message Passing Interface Forum. Mpi : A message-passing interface standard. *International Journal of Supercomputing Applications, 8(3/4)*, 1994.

[14] Benjamin Gaidioz, Rich Wolski, and Bernard Tourancheau. Synchronising network probes to avoid measurement intrusiveness with the network weather service. In *Proceedings 9th IEEE International Symposium on High Performance Distributed Computing. IEEE Computer society*, 2000.

[15] David Gauchard. *Simulation hybride des réseaux IP-DIFFSERV-MPLS multiservices sur environnement d'exécution distribuée.* PhD thesis, Université Paul Sabatier Toulouse, 2003.

[16] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Bob Manchek, and Vaidy Sunderam. *PVM : Parallel Virtual Machine - A User's Guide and Tutorial for Network Parallel Computing.* MIT Press, 1994.

[17] G.Franks, A. Hubbard, S. Majumdar, D.C. Petriu, J. Rolia, and C.M. Woodside. A toolset for performance engineering and software design of client-server systems. *Performance Evaluation*, 24(1-2) :117–135, 1995.

[18] G.Gayola, B.Lecointe, P.Bacquet, J.M.Parot, J.M.Garcia, T.Monteil, P.Pascal, and S.Richard. Casp : Clusters for application service provider. In *Journées 2002 Réseau National de Recherche et d'Innovation en Technologies Logicielles. Ateliers RNTL-ASTI Partenariat et Transfert de Technologie, Toulouse*, 2002.

[19] Global grid forum gridrpc-wg, https ://forge.gridforum.org/projects/gridrpc-wg/.

[20] M. Goldszmidt, D. Palma, and B. Sabata. On the quantification of e-business capacity. In *ACM Conference on Electronic Commerce (EC 2001)*, Florida, USA, October 2001.

[21] A.S. Grimshaw, Wm.A.Wulf, and the Legion team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 1997.

[22] Choi H. and Limb J. A behavioural model of web traffic. In *International conference of networking protocol 99 (ICNP 99).*

[23] R.L. Henderson. Job scheduling under the portable batch system. In *Job Scheduling Strategies for Parallel Processing. IPPS'95 Workshop.*

[24] Thomas F. HERBERT. *The LINUX TCP/IP STACK : Networking for Embedded Systems.* Programming Series. Charles River Media, Inc., 2004.

[25] Foster I. What is the grid ? a three point checklist. GRIDToday, 2002.

[26] Foster I. and Kesselman C. Globus : A metacomputing infrastructure toolkit. In *Intl J. Supercomputer Applications*, 1997.

[27] Foster I. and Kesselman C. The globus project : A status report. In *IPPS/SPDP '98 Heterogeneous Computing Workshop*, 1998.

[28] Foster I. and Kesselman C. *The Grid : Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann Publishers, inc., 1999.

[29] Foster I., Kesselman C., Nick J., and Tuecke S. The physiology of the grid : An open grid services architecture for distributed systems integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum., 2002.

[30] Site web de ibm cluster systems management, http ://www-03.ibm.com/servers/eserver/clusters/software/csm.html.

[31] Information Sciences Institute. Dod standard transmission control protocol. Technical report, University of Southern California, 1980.

[32] Information Sciences Institute. Transmission control protocol. Technical report, University of Southern California, 1981.

[33] Information Sciences Institute. Hypertext transfer protocol – http/1.0. Technical report, 1996.

[34] Information Sciences Institute. Hypertext transfer protocol – http/1.1. Technical report, 1999.

[35] Information Sciences Institute. The newreno modification to tcp's fast recovery algorithm. Technical report, 1999.

[36] Information Sciences Institute. Tcp congestion control. Technical report, 1999.

[37] Information Sciences Institute. The newreno modification to tcp's fast recovery algorithm. Technical report, 2004.

[38] Site web de la communauté jini, http ://www.jini.org.

[39] Seymour K., YarKhan A., Agrawal S., and Dongarra J. Netsolve : Grid enabling scientific computing environments. *Grid Computing and New Frontiers of High Performance Processing*, 2005.

[40] Humaira Kamal. Description of lam tcp rpi module. Technical report, University of British Columbia, Computer Science Department, 2004.

[41] "Krishna Kant". *"Introduction to Computer System Performance Evaluation"*. "McGraw-Hill, inc", 1992.

[42] L. Kleinrock. *Queueing Systems. Vol. 1 : Theory.* Wiley, 1975.

[43] Ferreira L., Berstis V., Armstrong J., Kendzierski M., Neukoetter A.and Takagi M., Bing-Wo R., Amir A., Murakawa R., Hernandez O., Magowan J., and Bieberstein N. Introduction to grid computing with globus. Redbook, IBM, 2002.

[44] Sing Li. *Professional JINI.* Wrox Press, 2000.

[45] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[46] Chetty M. and Buyya R. Weaving computational grids : How analogous are they with electrical grids ? *Computing in Science and Engineering*, 2002.

[47] Quinson M. *Découverte automatique des caractéristiques et capacités d'une plate-forme de calcul distribué.* PhD thesis, ENS de Lyon, 2003.

[48] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system : design, implementation, and experience. *Parallel Computing, volume 30, issue 7*, 2004.

[49] M.Soberman. *Les grilles informatiques.* Hermès Science, 2003.

[50] Desaulniers-Soucy N. and Iuoras A. Traffic modelling with universal multifractal. In *IEEE Globecom'99*, pages 1058–1065, 1999.

[51] Hidemoto Nakada, Mitsuhisa Sato, and Satoshi Sekiguchi. Design and implementations of ninf : towards a global computing infrastructure. *Future Generation Computing Systems*, 15 :649–658, 1999.

[52] Anand Natrajan, Anh Nguyen-Tuong, Marty A. Humphrey, and Andrew S. Grimshaw. The legion grid portal. Technical report, Legion Team, 2001.

[53] Nicolas NICLAUSSE. *Modélisation, Evaluation de Performances et Dimentionnement du World Wide Web.* PhD thesis, Université de Nice - Sophia Antipolis, 1999.

[54] Site web de ninf web, http ://ninf.etl.go.jp.

[55] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, and D.V. Wilcox. Pace - a toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3) :228–251, 2000.

[56] Site web d'opnet, http ://www.opnet.com.

[57] Barford P. and Crovella M. Generating representative workloads for network and server performance evaluation. In *ACM SIGMETRICS 98*, pages 151–160, 1998.

[58] Patricia PASCAL. *Gestion de ressources pour des services déportés sur des grappes d'ordinateurs avec qualité de service garantie.* PhD thesis, INSA Toulouse, 2004.

[59] P.Bacquet, J.M.Garcia, G.Gayola, T.Monteil, P.Pascal, and S.Richard. Projet casp, rapport final. Technical report, RNTL, 2003.

[60] P.Bacquet and S.Richard. Guide d'utilisation de netquad distribué. Livrable Projet RNTL CASP No23, Delta-Partners-Groupe Anite, Aout 2003.

[61] P.Pascal, S.Richard, and T.Monteil. Architecture of a grid resource manager. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 4, pages 2086–2092, 2002.

[62] Lock R. An introduction to the globus toolkit. Technical report, 2002.

[63] Site web de rocks cluster distribution, http ://www.rocksclusters.org.

[64] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. Softw. Eng.*, 21(8) :689–700, 1995.

[65] Tuecke S., Czajkowski K., Foster I., Frey J., Graham S., Kesselman C., Maguire T., Sandholm T., Vanderbilt P., and Snelling D. Open grid services infrastructure (ogsi) version 1.0. Draft recommendation, Global Grid Forum, 2003.

[66] Site web de sun grid engine, http ://www.gridengine.sunsource.net.

[67] Sun jini network technologie web site, http ://www.sun.com/software/jini/.

[68] Standard performance evaluation corporation.

[69] Daniel P. Spooner, Junwei Cao, Stephen A. Jarvis, Ligang He, and Graham R. Nudd. Performance-aware workflow management for grid computing. *Comput. J.*, 48(3) :347–357, 2005.

[70] Jeffrey M. Squyres, Brian Barrett, and Andrew Lumsdaine. Request progression interface (RPI) system services interface (SSI) modules for LAM/MPI. Technical Report TR579, Indiana University, Computer Science Department, 2003.

[71] Jeffrey M. Squyres and Andrew Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag.

[72] S.Richard. Interface d'accès pour l'interacteur. Livrable Projet RNTL CASP No23, Delta-Partners-Groupe Anite, LAAS-CNRS, Mars 2003.

[73] Site web de sun cluster, http ://www.sun.com/software/cluster/index.xml.

[74] Sun Microsystems, Inc. *N1 Grid Engine 6 User's Guide.*

[75] Sandholm T. and Gawor J. Globus toolkit 3 core - a grid service container framework. Technical report, Globus Alliance, 2003.

[76] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems : Principles and Paradigms*. Prentice Hall, 2002.

[77] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[78] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing : Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[79] Berstis V. Fundamentals of grid computing. Redpaper, IBM, 2002.

[80] Frost V. and Melamed B. Traffic modelling for telecommunication network. *IEEE Communication Magazine*, 3(32) :70–81, 1994.

[81] Matthew Wilcox. I'll do it later : Softirqs, tasklets, bottom halves, task queues, work queues and timers. In *Linux Conference, Australia, Perth*, January 2003.

[82] Rich Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings 6th IEEE Symposium on High Performance Distributed Computing, Portland Oregon.*, 1997.

[83] Rich Wolski, Neil Spring, and Jim Hayes. Predicting the cpu availability of time-shared unix systems on the computational grid. In *Proceedings 8th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society.*, 1997.

[84] S. ZHOU, J. WANG, X. ZHENG, and P. DELISLE. Lsf : Load sharing in large-scale heterogeneous distributed systems. In *Work- shop on Cluster Computing*, 1992.

Simulation et conception de services déportés sur grappes.

Résumé : Cette thèse étudie et propose des solutions à la problématique de mise en place de services déportés sur des grappes de machines. Ce mode d'utilisation de l'outil informatique permet de fournir la puissance de traitement sous forme de services.

Les problématiques d'observation de l'état des machines, de haute disponibilité, de prise en compte de l'aspect dynamique d'une grappe de machines et de gestion d'accès personnalisés aux grappes de machines sont étudiées. Le gestionnaire de ressources AROMA (scAlable ResOurces Manager and wAtcher), développé durant cette thèse permet d'apporter des réponses concrètes à ces problématiques. L'originalité de cet outils réside dans la prise en compte de l'aspect dynamique des grappes de machines, et dans la précision des informations d'état collectés sur les machines, ce qui autorise notamment la prise de décisions de placement intégrant la notion de qualité de service.

La problématique du dimensionnement des grappes de machines déportées est ensuite abordée. Différents modèles de simulations, intégrés au simulateur DHS (Distributed Hybrid Simulator) sont présentés. Les résultats obtenus par ce simulateur sont comparés à des modèles analytiques et à des mesures d'exécutions réelles afin de valider leur pertinence et d'évaluer les performances de la simulation. L'originalité de ce travail réside dans la simulation de l'ensemble des éléments ayant une influence sur les performances d'une grappe de machines : les clients, les applications, les machines, les systèmes d'exploitation et la couche réseau.
Mots-clés : A.S.P., grappes et grilles de calcul, simulation événementielle, gestion de ressources, évaluation de performances.

Application Service Provider environment conception and simulation.

Abstract : Due to Clusters and Grids popularity, the Application Service Provider concept, in which costumers pay for the use of software resources, is increasingly used.

The AROMA (scAlable ResOurces Manager and wAtcher) resource management system developed during this thesis studies the main problematic of service providers systems : load monitoring, high availability, hardware and software dynamicity and identity based level of service. AROMA originality both resides in its ability to deal with dynamic aspects of clusters in one hand, and in the accuracy of the state information collected on cluster nodes on the other hand, which makes it possible to deal with Quality of Service notion during the scheduling process.

This work also studies cluster resources dimensioning. Several simulation models, integrated into the Distributed Hybrid Simulator (DHS) are presented. Those models are validated and their efficiency evaluated by comparing simulation results to both theoretical analytic results and measurements. This work originality remains in the simulation of all the elements having an influence on the global system performances : customers, software, hardware, operating systems and network components. Proposed simulation models give several detail levels in order to select the appropriate precision/performance ratio.
Key-words : A.S.P, clusters and Grids, event driven simulation, resource management, performance evaluation.