Université de Montréal

Allocation optimale des ressources pour les applications et services de grille de calcul

par
Filali Abdelhanine

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maîtrise ès sciences (M.Sc.)
en Informatique

Avril, 2008

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Allocation optimale des ressources pour les applications et services de grille de calcul

présenté par :

Filali Abdelhanine

a été évalué par un jury composé des personnes suivantes :

Bernard Gendron, président

Abdelhakim Hafid, directeur de recherche

Michel Gendreau, codirecteur de recherche

Stefan Monnier, membre du jury

Résumé

Les applications utilisant l'infrastructure de grille de calcul nécessitent souvent l'allocation des ressources (par exemple, bande passante et CPU) pour la satisfaction de leurs besoins en qualité de service. Etant donné la nature dynamique de la grille de calcul, le support de la qualité de service et son adaptation sont d'une grande priorité pour supporter avec succès ces applications.

Une mauvaise réservation/allocation des ressources occasionne, suite au manque des ressources disponibles à un moment donné, des rejets des nouvelles demandes de ressources et par la suite des pertes de revenu substantielles aux fournisseurs des ressources/services. Deux contributions ont alors été proposées pour faire face à ce problème.

La première contribution qui concerne un type de ressources a pour objectif de minimiser le taux de rejet et de maximiser le revenu des fournisseurs avec satisfaction des besoins des utilisateurs. Et ceci, en donnant donner la possibilité aux clients d'exprimer plusieurs qualités de service pour un même type de ressources, grâce à la formulation d'un problème d'optimisation utilisant une approche à base de programmation linéaire. Deux heuristiques ont également été proposées pour la résolution, l'une qui sert à déterminer rapidement une bonne solution de départ et l'autre lancée en arrière-plan permettant d'améliorer la solution de départ. Notre approche reste applicable pour plusieurs ressources hétérogènes indépendantes. Il suffit de l'exécuter autant de fois qu'il y a de ressources demandées.

La deuxième contribution s'adresse tout particulièrement aux applications qui nécessitent deux types de ressources dépendantes, CPU et bande passante, et qui a les mêmes objectifs que la première contribution. Le problème est formulé comme un problème d'optimisation en utilisant une approche également à base de programmation linéaire, qui a pour fonction objectif la minimisation d'utilisation/coût des ressources réseau (bande passante). Notre idée derrière cette contribution est de minimiser la

probabilité de blocage dû au manque de bande passante et de minimiser par la suite le taux d'arrêt brusque des sessions dû aux pannes réseaux. Une heuristique composée de trois phases a été proposée pour la résolution; les deux premières phases servent à donner une bonne solution initiale et la troisième phase est une méthode tabou qui sert à améliorer la solution initiale.

Mots-clés : bande passante, qualité de service, programmation linéaire, heuristiques, contrat de niveau de service.

Abstract

Applications utilizing Grid computing infrastructure usually require resource allocation (e.g., bandwidth and CPU) to satisfy their Quality of Service (QoS) requirements. Given the dynamic nature of grid computing, QoS support and adaptation must be a high priority to successfully support these applications.

Poor reservation/allocation of resources causes, due to the shortage of resources available at a specific time, rejection of new requests and significant revenue losses to providers of resources and services. Two contributions have been proposed to deal with this problem.

The first contribution considers one type of resources and aims to minimize the rejection rate and maximize providers' revenues with satisfaction of users' requirements. And this, by providing the opportunity for clients to express more than one quality of service for the same type of resources, and by formulating the problem as an optimization problem using linear programming based approach. Two heuristics have been proposed for the resolution; the first one is used to quickly determine a good initial solution and the second one launched in background to improve the initial solution.

The second contribution is designed specifically for grid applications that require two types of dependents resources, namely CPU and bandwidth, and which has the same objectives as the first contribution. We formulated the problem as an optimization problem using also linear programming based approach, which has as an objective function that minimizes the use/cost of network resources. Our idea behind this contribution is to minimize the blocking probability and thereafter, to minimize the rate of sessions termination due to network failures. An heuristic that consists of three phases has been proposed for solving the problem; the two first phases are used to quickly determine a good initial solution, and the third one is a tabu search that aims to improve the initial solution.

Keywords: Bandwidth, Quality of service, Integer programming, Heuristics, Service Level Agreement.

Table des matières

Liste des tableaux

Liste des figures

Chapitre1

Chapitre 2

Chapitre 3

Chapitre 4

Liste des sigles et des abréviations

Acronyme	Description
QoS	Quality of Service
QdS	Qualité de Service
IP	Integer Programming
BIP	Binary Integer Programming
LOH	Local Optimization Heuristic
GOH	Global Optimization Heuristic
GI	Greedy Insertion
GIT	Greedy Insertion plus Tabu
SLA	Service Level Agreement
WAITLIST	WAITING LIST
LSP	Labeled Switch Path
Mbps	Mega bit per second
NSEC	Neighbourhood Structure Based on Ejection Chains

À mes parents.

À mon épouse.

À mes enfants.

À toute ma famille.

À toute ma belle-famille

Remerciements

Je tiens tout d'abord à adresser mes remerciements et ma gratitude à mon directeur et mon codirecteur de recherche, les professeurs Abdelhakim Hafid et Michel Gendreau pour leur patience, leur disponibilité, leur aide et leur support tout au long de ce travail.

J'aimerais remercier ma femme Fadoua pour sa présence et son soutien permanents ainsi que mes enfants Aya et Houda pour leur patience et leur sagesse.

Enfin, que tous celles et ceux qui m'ont apporté leur appui trouvent ici l'expression de mes sincères remerciements

Chapitre 1

Introduction

1.1 Grilles de calcul

Depuis quelques années, l'informatique répartie et les technologies associées sont dans une constante évolution technologique et économique. Les technologies sont de plus en plus matures et sophistiquées. Les serveurs de calcul et de stockage voient leur rapport qualité/prix en constante augmentation en intégrant les nouvelles innovations technologiques et de fabrication. La même observation peut être appliquée aux technologies réseaux et à la bande passante offerte qui tend à devenir illimitée d'un point de vue applicatif.

La réponse à ces changements est de passer à un modèle d'informatique répartie permettant d'exploiter pleinement les ressources et les capacités offertes. Cet environnement offrira un service et un accès uniforme et économiquement viables aux ressources de l'infrastructure. Cette évolution est connue sous le nom de «Grid Computing» ou « Grilles de Calcul ».

Le terme « The Grid » ou « grille de calcul », a été introduit pour la première fois aux Etats-Unis durant les années 1990 pour décrire une infrastructure de calcul répartie utilisée dans des projets de recherche scientifique et industrielle.

Il est important de savoir quels avantages une grille de calcul est en mesure d'offrir que les infrastructures et les technologies actuelles ne sont pas capables d'assurer. Nous exposons par la suite quelques-unes des raisons pouvant amener à déployer une grille de calcul.

- **Exploiter les ressources sous-utilisées:** Les études montrent que les ordinateurs personnels et les stations de travail sont inactifs la plupart du temps.

- **Fournir une importante capacité de calcul parallèle:** Le milieu industriel bénéficiera énormément d'une telle capacité.
- **Meilleure utilisation de certaines ressources** : Le partage des ressources permet l'accès à des ressources spéciales comme des équipements spécifiques (microscope électronique, bras robotique ...) ou des logiciels dont le prix de la licence est élevé.
- **Fiabilité et disponibilité des services** : La grille de calcul assure la continuité du service si certaines ressources deviennent inaccessibles. Les logiciels de contrôle et de gestion de la grille seront en mesure de soumettre la demande de calcul à d'autres ressources.

Les principales applications des grilles de calcul se feront dans les domaines suivants :

- **Supercalculateur réparti** ("Distributed Supercomputing") : Une grille de calcul pourra agréger une importante quantité de ressources afin de fournir la puissance de calcul nécessaire pour de nombreuses applications. Comme exemples d'applications nous pouvons citer la simulation distribuée dans la météorologie, la cosmologie, etc.
- **Calcul haut-débit** ("High-Throughput Computing") : Une grille de calcul sera utilisée pour ordonnancer en parallèle une importante quantité de tâches indépendantes les unes des autres. Comme exemples d'applications nous pouvons citer la recherche de clés cryptographiques, les simulations de molécules et l'analyse du génome ...
- **Calcul sur demande** ("On-Demand Computing") : Une grille de calcul pourra fournir les ressources pour satisfaire les demandes à court terme d'une application que les ressources locales ne sont pas en mesure d'assurer (cycles processeur, stockage ...). Le défi principal pour les concepteurs de telles grilles est la nature dynamique et aléatoire des demandes faites par les utilisateurs qui peuvent constituer une large population.

- **Calcul Collaboratif** ("Collaborative Computing") : Cette classe d'applications inclut les applications d'interaction entre humains dans des environnements de simulation en temps-réel (conception et interaction avec un nouveau moteur d'avion, etc.).
- **Génération, traitement et stockage d'énormes quantités de données** ("Dataintensive Computing") : Dans de telles applications, une grille de calcul pourra absorber et stocker une importante quantité d'informations générées. Comme exemple d'applications nous pouvons mentionner la production d'une carte de l'univers, la prévision météorologique à long terme, les simulations quantiques ...

Les grilles de calcul possèdent quatre principales caractéristiques:

- **Existence de plusieurs domaines administratifs** : Les ressources sont géographiquement distribuées et appartiennent à différentes organisations chacune ayant ses propres politiques de gestion et de sécurité.
- **Hétérogénéité des ressources** : Les ressources sont de nature hétérogène en termes de matériels et de logiciels.
- **Passage à l'échelle** ("Scalability") : Une grille pourra consister de quelques dizaines de ressources à des millions. Cela pose de nouvelles contraintes sur les applications et les algorithmes de gestion des ressources.
- **Nature dynamique des ressources** : Cela pose des contraintes sur les applications telles que l'adaptation au changement dynamique du nombre de ressources, la tolérance aux pannes et aux délais ...

Les ressources constituant la grille sont en général réparties en plusieurs types, cycles processeur, capacité de stockage, bande passante, équipements spéciaux et logiciels à licence élevée, etc. La figure ci-dessous illustre une telle architecture:

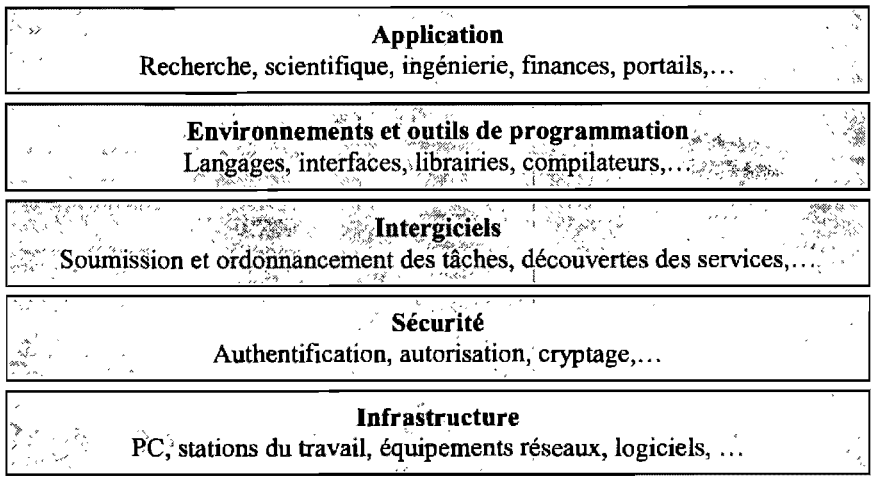

Figure 1.1 : Architecture générale d'une grille de calcul.

Dans la couche la plus basse de l'architecture, en cinq couches, nous avons les ressources proprement dites interconnectées à travers des réseaux locaux et grandes distances. Cette infrastructure matérielle ("fabric") comprend notamment des PC, des stations de travail, des grappes de calcul ("clusters"), des équipements de stockage, des bases de données, des équipements spéciaux, etc. La seconde couche fournit les mécanismes de sécurité nécessaires à la protection des ressources. La troisième couche fournit les intergiciels ("middlewares") nécessaires à la gestion des ressources, la coordination de l'accès, l'ordonnancement des tâches, etc.

Nous expliquons ci-dessous brièvement les principales fonctions assurées par ces intergiciels :

- **Ordonnancement** ("Scheduling") **:** Un ordonnanceur ("scheduler") devra trouver les ressources les plus appropriées pour faire tourner les tâches qui lui sont soumises. Ils peuvent aussi superviser le déroulement d'une tâche jusqu'à sa terminaison, la soumettre à nouveau si elle est brusquement interrompue et la terminer prématurément si elle se trouve dans une boucle infinie d'exécution.
- **Réservation :** La réservation permet la garantie de la qualité de service, ainsi que le respect de certaines échéances. Pour cela l'intergiciel devra fournir un système de

réservation permettant aux utilisateurs d'exprimer leurs besoins en ressources et d'effectuer la réservation.

- **Services d'information et d'annuaire :** Ce sont des mécanismes permettant d'enregistrer et de découvrir les ressources et d'identifier leur état.
- **Service de nom** ("Naming Service") **:** Le service permet un accès transparent aux ressources en cachant toutes les différences « physiques ».

La quatrième couche regroupe tous les outils et les paradigmes pouvant aider les développeurs à concevoir et écrire des applications pouvant tourner sur la grille. On y trouve plus particulièrement des API (découverte et réservation de ressources, mécanismes de sécurité, stockage ...) que les développeurs d'applications pourront utiliser.

La dernière couche regroupe les applications proprement dites qui sont de nature variée : projets scientifiques, médicaux, financiers, ingénierie, etc.

1.2 Motivation et problématique

La réservation / allocation des ressources est une tâche du gestionnaire de ressource qui fait partie de la troisième couche de l'architecture en couche de la grille de calcul (Figure 1.1). Étant donné que la plupart les applications utilisant la grille de calcul ont la contrainte de temps, le soutien et l'assurance en qualité de service pour s'exécuter correctement est un besoin primordial. Il est donc impératif d'apporter des garanties de qualité de service plus rigoureuses au-delà de celles prévues par la grille des infrastructures de base.

Suite à de mauvaises réservations / allocations de ressources, des demandes de services / ressources pourraient être rejetées et par la suite le revenu des fournisseurs sera influencé. Par exemple, étant donné 3 processeurs C1, C2 et C3 de capacité 100 chacun et 3 demandes (demande A de 60 cycles, demande B de 40 cycles, et demande C de 50), on affecte la demande A à C1, la demande B à C2, et la demande C à C3. Quand une demande

D qui nécessite 70 cycles est reçue, elle est donc rejetée du fait qu'il n'y a aucun processeur qui a au moins 70 cycles disponibles. Il aurait été possible de l'accepter si A et B avaient été affectées à C1. Un autre exemple de mauvaise utilisation de ressources est d'affecter une demande à deux processeurs qui sont localisés dans des domaines différents (et ont donc besoin de bande passante pour communiquer), alors que c'est possible de l'affecter à 3 processeurs (de moindre capacité) localisés dans le même domaine.

De ce fait, la gestion optimale des ressources est une préoccupation vitale dans la gestion de la grille de calcul, qui pourrait aider à la satisfaction des utilisateurs avec le respect des contrats de niveau de service durant l'exécution des tâches. Nous entendons par optimale, une façon qui diminue le taux de blocage avec la satisfaction des demandes d'une part, et qui minimise le coût et/ou qui maximise les revenus d'utilisation des ressources d'autre part.

Au vu de l'importance de réservation / allocation efficace des ressources plusieurs approches ont été proposées. La plupart de ces approches existantes, sinon toutes, considèrent la spécification par l'utilisateur, au moment de la demande, d'une valeur de qualité de service pour chaque type de ressources. Ceci a une conséquence directe sur l'augmentation du taux de rejet et par la suite sur la diminution du revenu des fournisseurs. Par exemple, au moment « t » une application « A » a besoin de 20 cycles de CPU durant 1 heure et la capacité disponible en ce temps-là est de 15 cycles et après 5 minutes la capacité disponible sera de 50 cycles. Au lieu de rejeter cette application, comme c'est le cas dans plusieurs travaux, on pourrait l'accepter avec 15 cycles et après 5 minutes on lui affecte 20 cycles. En outre, plusieurs applications ont besoins de plusieurs types de ressources dépendantes. Or, la plupart de ces approches ne tiennent pas compte de cette dépendance même si elles considèrent les ressources hétérogènes.

1.3 Contributions

Dans ce projet, nous avons considéré la réservation / allocation de ressources dans la grille de calcul. Nous nous sommes particulièrement intéressés à la résolution des problèmes cités précédemment concernant le blocage des demandes et le revenu / coût des fournisseurs de services. Dans ce sens, nous avons proposé deux contributions.

La première contribution consiste à donner aux clients la possibilité de spécifier plus d'une qualité de service pour un type de ressources déterminé. Le client est supposé satisfait si on lui attribue le minimum des qualités spécifiées. Ceci donne naturellement plus de chance à l'utilisateur d'être accepté et par la suite à la réduction du taux de blocage, et donne, en outre, plus de choix également aux fournisseurs de choisir la réservation / allocation optimale des ressources. Nous avons proposé un modèle d'optimisation qui maximise les revenus des fournisseurs et ainsi minimise le taux de blocage tout en satisfaisant les exigences des utilisateurs. Nous avons proposé également dans le même contexte deux heuristiques qui permettent de résoudre le modèle: (i) LOH, utilisée pour donner une bonne solution initiale en un temps de réponse acceptable; et (ii) GOH, une heuristique qui se lance périodiquement en arrière-plan pour l'amélioration de la solution courante.

La deuxième contribution concerne l'allocation de deux types de ressources qui sont dépendantes. Pour ce problème, nous avons proposé, spécialement pour les applications de grille de calcul qui ont besoin simultanément de deux types de ressources dépendantes (CPU et bande passante), une approche qui permet de minimiser l'utilisation de la bande passante avec satisfaction des demandes. Et ceci, par la formulation du problème comme un problème d'optimisation en utilisant une approche à base de programmation linéaire, avec la proposition d'une heuristique composée de trois phases pour la résolution : (i) les deux premières phases déterminent la bonne solution de départ ; et (ii) la troisième phase, une méthode tabou, améliore la solution de départ. Notre idée principale derrière cette

contribution est de minimiser la probabilité de blocage dû au manque de bande passante et de minimiser par la suite le taux d'arrêt brusque des sessions dû aux pannes réseaux.

Ce mémoire est présenté sous la forme par articles. Le premier article « Adaptive Resources Provisioning for Grid Applications and Services » [7], a été accepté dans IEEE ICC, 2008. Le second article « Bandwidth and Computing Resources Provisioning for Grid Applications and Services » [8], a été soumis à IEEE GC, 2008.

Les idées et les objectifs principaux derrière ces contributions ont été proposés par le directeur de recherche A. Hafid. L'étude approfondie de ces idées et de ces objectifs a été réalisée par l'étudiant, avec la proposition des modèles d'optimisation et des méthodes de résolution. Les modèles d'optimisation, ainsi que les méthodes de résolution, ont été validés par le co-directeur de recherche M. Gendreau. Ensuite, l'implémentation a été développée par l'étudiant. Les tests visant à valider cette implémentation, ont été guidés par le directeur de recherche A. Hafid.

1.4 Organisation du mémoire

Le reste de ce mémoire est organisé comme suit. Dans le chapitre 2, nous présentons un aperçu de la littérature sur les approches utilisées pour résoudre spécialement le problème du blocage et qui abordent le revenu et/ou coût d'utilisation des ressources dans le domaine de grille de calcul. Dans le chapitre 3, nous présentons en format article notre première contribution qui permet de réduire le taux de blocage et de maximiser le revenu des fournisseurs pour un type de ressources. Dans le chapitre 4, nous proposons en format article notre seconde contribution qui a les mêmes objectifs que la première contribution, mais avec deux types de ressources dépendantes (CPU et bande passante). Finalement, nous concluons ce travail en dressant quelques perspectives de recherche.

Chapitre 2

Revue de littérature

La gestion des ressources et de la Qualité de Service (QdS) ont été étudiées dans différents contextes, en particulier pour les réseaux informatiques et les applications multimédia. Elles ont également été étudiées dans le contexte de la grille de calcul. Quel que soit le contexte, un système de gestion de la QdS doit porter sur la spécification des besoins de QdS, la projection des besoins de QdS aux capacités des ressources « mapping », la négociation de QdS avec les propriétaires des ressources quand les besoins de QdS ne peuvent être réalisables complètement, l'établissement des contrats de niveau de service avec les clients, la réservation et l'allocation des ressources, la surveillance des paramètres de QdS associés à la session, l'adaptation aux variations des états de ressources et les opérations de fin des sessions.

Beaucoup de travaux ont été effectués dans le contexte de la gestion des ressources et de la QdeS. Nous passons en revue les travaux les plus pertinents et qui sont en rapport avec notre travail. Les articles étudiés sont ceux qui traitent du problème de blocage des demandes, du problème d'optimisation d'allocation de ressources, spécialement, en termes de revenu et de coût d'utilisation des ressources. Nous les avons divisés selon qu'ils couvrent le problème avec un seul type de ressources ou avec plusieurs types de ressources. Un résumé de ces articles clôturera le chapitre en section 2.3.

2.1 Allocation d'une ressource

Les articles étudiés dans cette section dans le cadre de cette revue de la littérature sont parmi les plus pertinents, étant donné l'objectif de ce mémoire. Ils font état des

travaux réalisés pour résoudre les problèmes d'allocation de ressources cités dans l'introduction pour un type de ressources (CPU, bande passante, mémoire, etc.)

Les auteurs dans **[19]**, partent du principe que la bonne conception du contrat de niveau de service mène à la réduction du blocage des demandes et à la bonne optimisation d'allocation des ressources en termes de profit et de coût ; ils ont présenté une définition de service intégrée et générique pour la réservation des ressources sans distinction entre réservations immédiates et à l'avance. Ils rappellent particulièrement le problème de séparation conceptuelle des réservations immédiates et en avance, et le problème issu de la nécessité de spécifier la durée de réservation préalable, ce qui impose des restrictions sur le potentiel des demandes de service. Pour faire face à ces problèmes, les auteurs incluent dans leur définition de service la durée de non interruption de service, qui peut être aléatoire. La demande de réservation R est décrite au moment de la demande par (r: temps où la demande de réservation est faite, s: temps de début de service, e: temps de fin de service non interruptible, v: quantité de ressource demandée). Une information clé additionnelle pour cette description de service est qu'en tout temps t, chaque service est en état $p(t)$ (priorité d'interruption), avec : (i) $p(t)=1$ si $s \le t \prec e$, signifie que la demande de réservation est garantie de ne pas être interrompue pendant toute la période $[s,e]$; et (ii) $p(t)=0$ sinon, signifie que la demande est interruptible. La définition de service est schématisée dans la Figure 2.1.

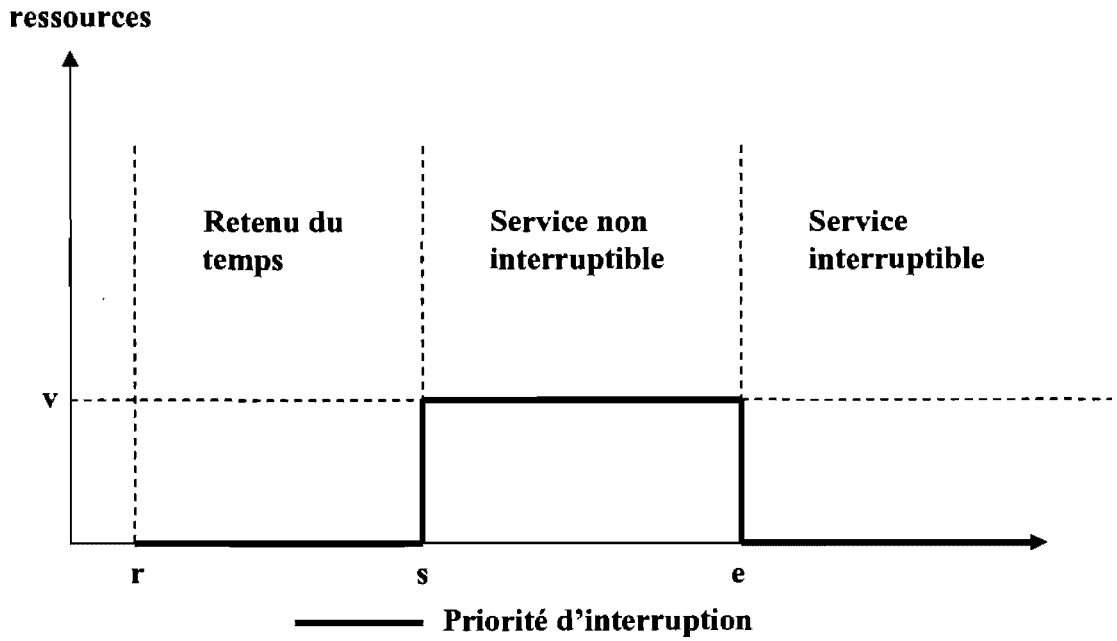

Figure 2.1 : Définition de service

Les auteurs ont proposé également un algorithme de contrôle d'admission qui considère uniquement les demandes non interruptibles. Ceci pour assurer la qualité de service pendant la période de non interruption et permettre aux nouvelles demandes d'être acceptées. Dans ce sens, la charge totale au temps t est définie comme suit:

$$Load(t) = \sum_{i=1}^{n} v_i p_i(t) \quad \text{Pour } p_i, v_i \text{ associés aux demandes R}i,\ i = 1, \cdots, n.$$

La contrainte de capacité est définie comme suit :

$$Load(t) \leq C \ Pour \ tout \ t, t \geq t_0$$

Avec ce contrôle d'admission les réservations nouvelles et existantes sont schématisées dans la Figure 2.2.

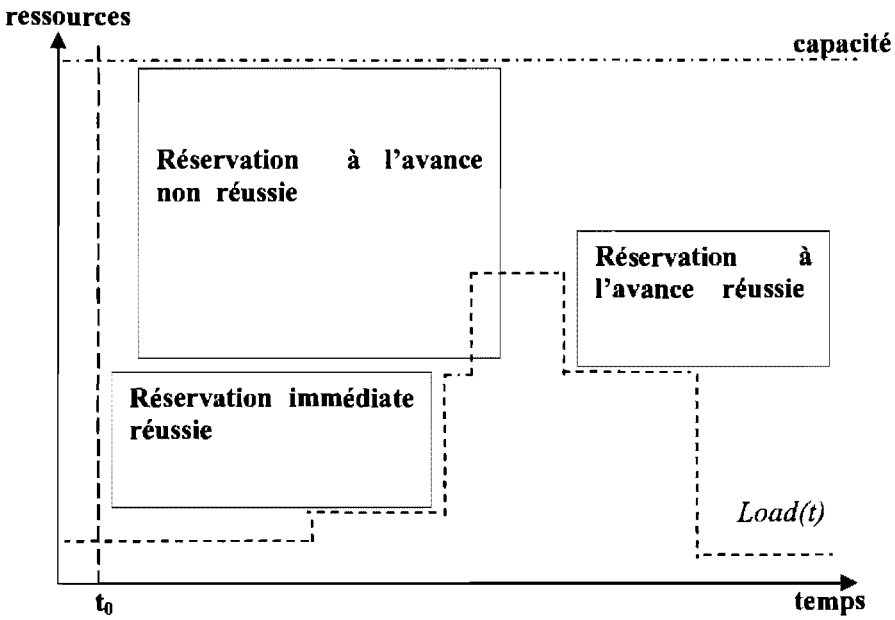

Figure 2.2 : Réservations existantes et nouvelles

Les auteurs dans **[20]** ont proposé un modèle de sponsorisation de ressource « Resource Broker » (RB), dans le contexte du middleware pour l'application de DMM « Distributed Multi-Media ». Le RB représente un gestionnaire intermédiaire de ressources entre le client et l'ordonnanceur de ressources (RS), qui supporte la réservation immédiate et à l'avance, avec la séparation des opérations de la sponsorisation des opérations de l'ordonnancement. Ceci permet d'obtenir un très bon temps de réponse en ce qui concerne les opérations de sponsorisations telles que la réservation et la libération des ressources. A travers le graphe de réservation qui indique les réservations des ressources effectuées dans des intervalles de temps spécifiques, le RB peut décider d'accepter la réservation, ou de proposer une autre qualité de service si la réservation avec la qualité de service spécifiée ne peut être acceptée, ou de proposer une autre durée de service inferieure à celle indiquée dans la demande ou encore de proposer un autre début de réservation. Le contrôle d'admission ainsi que le graphe de réservation sont schématisés dans les Figures 2.3. et 2.4.

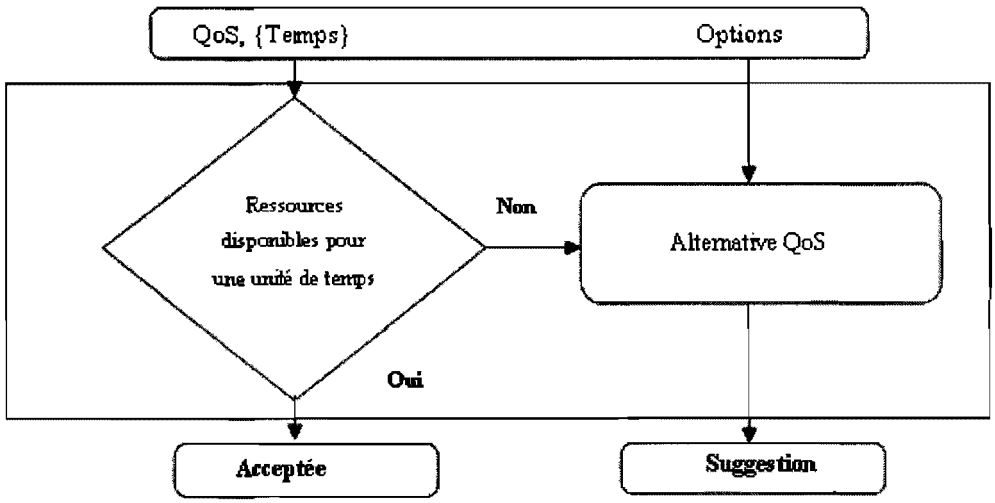

Figure 2.3 : Module de contrôle d'admission pour le service intégré

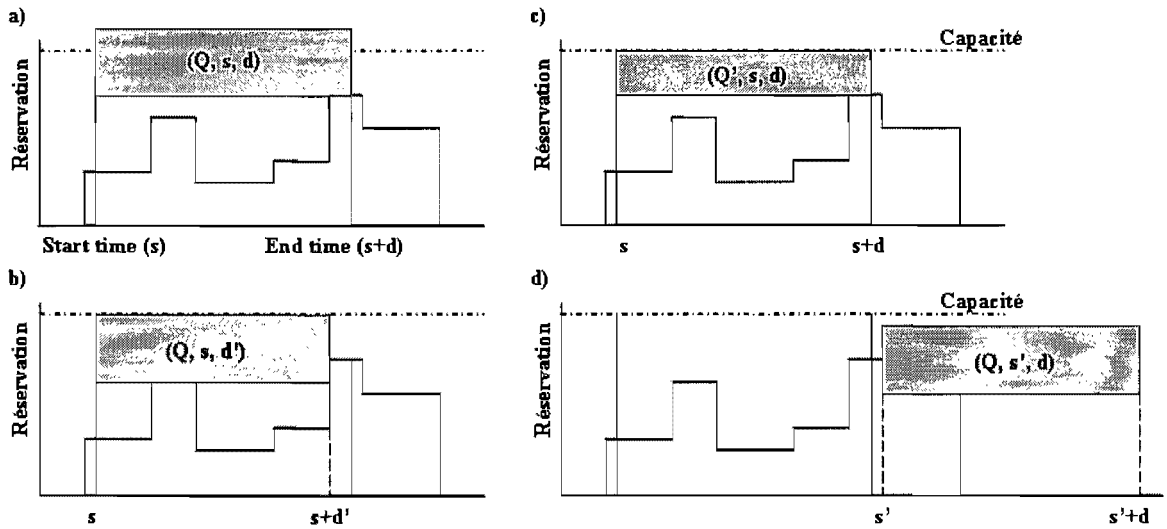

Figure 2.4 : Rejet de la demande de réservation (a) et suggestions alternatives (b,c,d).
Q : QdS demandée, s : temps de début, et d : durée.

L'implémentation du graphe de réservation est faite en utilisant l'arbre d'état adaptif avec le temps ("Timely Adaptive State Tree").

Les auteurs dans [3] ont traité le problème de la sélection d'une ressource qui correspond exactement aux spécifications introduites par l'utilisateur. L'objectif visé est de

trouver l'ensemble des ressources satisfaisant mieux la requête, étant donné que la requête mentionne *n* attributs de ressources voulues, et qu'on a une fonction d'évaluation qui a ces attributs comme entrée. La ressource qui retourne la plus grande valeur de la fonction est considérée comme une qui correspond mieux à la requête. Les auteurs mentionnent que les fournisseurs, les demandeurs et les services de sélection doivent partager la même structure de l'ontologie. Ils ont défini trois types de fonctions de sélection de ressources : (i) une fonction qui peut contenir les expressions booléennes et retourne une valeur booléenne ; (ii) une fonction qui peut contenir des expressions arithmétiques et retourne un réel positif ; (iii) une fonction qui peut contenir des expressions floues ("fuzzy expressions") et retourne une valeur floue. Le service de sélection exécute l'algorithme de sélection pour chaque requête qui se présente. Deux algorithmes de sélections ont été présentés, l'un qui permet de faire une recherche exhaustive et l'autre qui permet une recherche restrictive. Le travail tel qu'il est décrit ne présente aucune optimisation en termes d'utilisation de ressources.

Dans **[11]**, les auteurs ont présenté un travail qui concerne spécialement les applications réseau, qui ont besoin de réaliser une performance de bout en bout fiable; telles que visualisation à distance, télé-immersion, etc. Généralement les applications de ce genre optent pour la réservation ou pour l'adaptation de la qualité de service. Les applications utilisant la réservation spécifient la QdeS qui ne devrait pas changer durant toute la session (sauf en cas de panne ou d'interruption). Alors que les applications utilisant le service d'adaptation s'adaptent aux conditions réseau. La réservation a comme limite ce qu'elle ne permet pas la réduction de qualité de service, ce qui entraîne dans certaines situations l'interruption totale de l'application. La technique d'adaptation n'échappe pas de problème également, surtout lorsqu'il s'agit d'une forte congestion qui mène à une dégradation de qualité de service inacceptable. De ce fait, les auteurs ont présenté une approche de QdeS qui combine les caractéristiques de réservation et de l'adaptation, et qui a comme but de minimiser les limites citées ci-dessus. En fait, l'approche est composée de trois mécanismes : (i) Actuateurs, qui permettent aux applications de modifier leurs

réservations dynamiquement ; (ii) Capteurs, qui permettent aux applications de détecter que l'adaptation est nécessaire ; (iii) Procédures de décision, qui permettent d'invoquer les actuateurs en réponse aux informations des capteurs. L'expérimentation a montré que les applications avancées du réseau peuvent bénéficier de ces mécanismes. Ces mécanismes sont implémentés comme extension du système GARA (« Globus Architecture for Reservation and Allocation ») **[29]**.

Alors que les travaux précédents traitent le problème avec un certain nombre de ressemblances en termes d'objectifs et de spécifications, l'article **[23]** l'aborde comme un problème d'optimisation avec souci d'avoir une solution approchée dans un temps de réponse acceptable. La fonction objectif considérée est la maximisation du nombre de réservations réussies. Les auteurs dans cet article proposent une méthode hybride pour la résolution des problèmes d'optimisation sous contraintes en contexte tout temps (« *anytime* »). La méthode a été appliquée avec succès pour résoudre un problème de réservation de connexions dans les réseaux ATM: 49% des demandes qui auraient été rejetées sont désormais acceptées. Les méthodes de recherche complètes sont souvent utilisées pour obtenir des solutions optimales. Mais, du fait de leur comportement (souvent) exponentiel, ces méthodes peuvent s'avérer trop gourmandes en temps de calcul. De plus, il a été observé qu'en raison de leur caractère systématique d'exploration de l'espace de recherche, les solutions intermédiaires qu'elles produisent sont souvent de piètre qualité. Grâce à leur façon opportuniste d'explorer l'espace de recherche, les méthodes approchées, basées sur les recherches locales (comme le recuit simulé ou les recherches tabou), sont supposées produire de bonnes solutions en des temps de calcul abordables. Malheureusement, de telles méthodes ne sont pas toujours capables de se sortir facilement d'optima locaux et peuvent perdre beaucoup de temps dans l'exploration de voisinages inintéressants. Un tel comportement est prohibé dans un contexte « anytime», puisque la qualité des solutions doit s'améliorer graduellement, et le plus rapidement possible, au cours du temps. Les auteurs proposent dans ce contexte un algorithme hybride qui offre un réel compromis entre ces deux types d'approches. L'algorithme part d'une solution initiale

s; lors de chaque mouvement, il relaxe une partie de la solution courante, puis la reconstruit en sélectionnant la meilleure solution voisine de *s*. L'algorithme s'arrête dès que le nombre maximum de mouvements locaux autorisés (MAXMOVES) est atteint.

2.2 Co-allocation des ressources hétérogènes

Alors que tous les travaux présentés dans la section précédente considèrent un type des ressources, dans cette section nous allons considérer les travaux qui ont abordé le problème avec des ressources hétérogènes. Nous entendons par hétérogène, une combinaison de deux ou plusieurs types de ressources (CPU, mémoire, unités de stockage...). Les articles sélectionnés demeurent les plus pertinents en ce qui concerne l'objectif du travail dans ce mémoire.

Etant donné que chaque utilisateur de la grille de calcul demande une qualité de service particulière pour chaque type de ressources spécifique, les auteurs dans **[1]** ont élaboré un modèle de contrat de niveau de service, qui mentionne entre autres la classe de service demandée, que les fournisseurs doivent respecter durant toute la session d'exécution et dont les clients doivent payer les coûts associés. Ils ont formulé le problème comme un problème d'optimisation, qui a pour fonction objective de maximiser le revenu monétaire des services fournis, comme suit :

$$\text{Profit total} = \max \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} a_{ij} ,$$

où c_{ij} est le prix unitaire pour utiliser la ressource *j,* qui dépend de la classe de service mentionnée dans le contrat de service conclu avec le client *i*, et a_{ij} est la quantité de ressource *j* demandée par le client *i.* Les auteurs n'ont cependant pas présenté les détails du modèle et des contraintes, ni de solutions. Ils ont proposé un algorithme d'adaptation qui s'intéresse aux classes de services. Le client avec la classe de service élevée reçoit plus de priorité. En fait, l'algorithme d'adaptation proposé permet la réservation des ressources pour trois types de services : garanti (« guaranteed)»), meilleur service (« best effort»), et

adaptatif («adaptive»). L'algorithme essaie de répondre à chaque nouvelle demande par l'ajustement des réservations de ressources entre les trois types de services (par exemple, en réduisant la quantité de ressources réservée aux demandes de "meilleur service" pour pouvoir accepter une demande avec la qualité de service garantie). L'algorithme réserve un minimum de ressources aux demandes de classe « meilleur service ». L'ajustement tel qu'il est utilisé dans ce travail permet : (i) d'éviter la sous-utilisation des ressources de la grille; (ii) de réserver un minimum de capacité de ressources pour les clients de la classe meilleur service; et (iii) de réduire le taux de blocage.

Dans l'article **[2]**, les auteurs ont présenté un modèle de qualité de service qui fait intervenir le fournisseur de service, le service et les ressources, sous les hypothèses suivantes: (i) un fournisseur de service peut offrir un ou plusieurs services ; (ii) un service peut utiliser un ou plusieurs ressources pour s'exécuter ; (iii) les ressources peuvent être de différents types (réseau, CPU, disque,...). Les relations entre ces trois composants sont exprimées comme suit : $P \Delta \{S_1, \cdots, S_n\} \Delta \{R_1, \cdots, R_n\}$. Le travail possède beaucoup de ressemblances avec celui présenté dans **[1]**. Parmi les similarités, on note les suivantes :

- Conception d'un modèle de contrat de niveau de service, qui pourrait servir autant pour les réservations immédiates que pour les réservations à l'avance.
- Formulation du problème comme un problème d'optimisation en utilisant la programmation en nombres entiers avec la même fonction objectif qui vise à maximiser le profit des fournisseurs.

Les auteurs dans cet article n'ont pas proposé une méthode de résolution. Cependant, ils ont présenté une fonction de contrôle d'admission classique qui permet en un temps t de comparer la quantité disponible de ressource avec la quantité demandée. La demande est acceptée si toutes les quantités demandées de différentes ressources sont inferieures respectivement aux quantités disponibles de ces ressources. Autrement, la demande est rejetée.

Les auteurs dans **[25]** considèrent les co-réservations/co-allocations en ligne des ressources aux tâches de la grille de calcul exigeant plusieurs ressources hétérogènes. Les tâches considérées nécessitent une instance pour chaque type de ressources. Ceci exclut les tâches parallèles nécessitant plusieurs multiprocesseurs en même temps. Le problème est modélisé comme un problème d'optimisation en utilisant la programmation en nombres entiers, où la fonction objectif considérée dans l'implémentation est la maximisation du nombre de réservations immédiates réussies. Une tâche est considérée satisfaite si tous ses besoins en ressources sont satisfaits ; elle est exprimée comme un « Gang matching expression». La méthode de résolution utilisée pour le problème modélisé dans ce travail est une méthode de résolution exacte utilisant l'outil CPLEX.

Quant aux articles **[24]** et **[12]**, les auteurs traitent le même sujet qu'en **[25]**, mais avec des approches différentes. Dans **[24]**, le modèle d'allocation de ressources est basé principalement sur la notion « qualification de collection des ressources» *(QRC)* qui représente le minimum des ressources nécessaire pour satisfaire une tâche. Les auteurs ont supposé un certain nombre de règles pour l'affectation d'une tâche : (i) Une collection de ressources peut servir plusieurs tâches; (ii) Une tâche peut avoir plus d'un choix d'affectation aux *QRCs*; (iii) Une ressource peut être incluse dans plusieurs *QRCs*. Ils ont utilisé un algorithme évolutionnaire comme méthode de résolution, avec la proposition de trois méthodes pour construire une population de solutions initiales : (i) La première méthode est basée sur une construction des solutions initiales d'une façon aléatoire et indépendante des unes des autres comme dans l'algorithme évolutionnaire de base; (ii) La deuxième méthode, basée sur la méthode gloutonne (« greedy method ») procède en affectant les tâches suivant leurs priorités d'une façon décroissante ; si une tâche ne peut pas être affectée, elle est ignorée, et le processus recommence avec une autre tâche; (iii) La troisième méthode est une amélioration de la deuxième méthode ; elle permet, dans le cas où la tâche courante ne peut pas être affectée, de revenir en arrière et de réassigner aux QRCs alternatifs les tâches déjà affectées. Le processus de retour en arrière s'arrête quand

la limite de retour est atteinte ou bien la tâche courante peut être affectée. La tâche est ignorée si elle n'a pas pu être affectée, et le processus recommence avec une autre tâche.

Une fois la population initiale des solutions donnée, le processus de mutation commence. Il est utilisé pour créer une nouvelle solution basée sur une solution existante. Il commence chaque fois par la sélection d'une petite fraction de tâches affectées aux QRCs dans une solution existante en fonction d'une distribution aléatoire basée sur la faible priorité des tâches. Puis il supprime, de l'ensemble des tâches sélectionnées, des tâches selon la probabilité de mutation prédéfinie. Ceci libère quelques capacités de ressources pour pouvoir insérer les autres tâches. L'insertion se poursuit jusqu'à ce qu'il n'y a plus d'affectations possibles en appliquant les deux règles suivantes: (i) Les tâches à affecter sont sélectionnées sur la base de critère de priorité en ordre décroissant ; (ii) L'affectation est faite d'une façon aléatoire au QRC en respectant les contraintes de capacité. À la fin de chaque itération du processus de mutation, une nouvelle solution est créée. Le processus s'arrête quand l'une des conditions prédéfinies d'arrêt se réalise. Finalement, la meilleure solution trouvée est donnée comme solution du problème.

Quant à l'article **[12]**, le problème est abordé avec beaucoup de ressemblances avec l'article **[25]**, sauf que les auteurs en **[12]** ont introduit la notion de la ressource virtuelle qui est associée au plus à une ressource physique. Brièvement, voici quelques caractéristiques du problème abordé : (i) une ressource virtuelle est associée au plus à une ressource physique ; (ii) une ressource physique peut être associée à zéro ou plusieurs ressources virtuelles ; (iii) un service est déployé si tous ses besoins sont satisfaits par les ressources virtuelles affectées ; (iv) Pour chaque ressource virtuelle (RV) et chaque attribut dynamique de la ressource physique où RV est déployée, un attribut virtuel est défini avec la capacité égale à la capacité disponible de l'attribut physique. La modélisation du problème est faite avec la programmation en nombres entiers qui tient compte des ressources virtuelles et des règles citées ci-dessus. L'approche heuristique pour la résolution est exécutée en deux phases. La première phase consiste à sélectionner pour

chaque service les ressources virtuelles où il sera déployé. Alors que l'objectif de la seconde phase est d'associer les ressources virtuelles, nouvellement créées par la première phase, aux ressources physiques. Plus précisément, dans la première phase, l'heuristique considère juste les services non encore affectés à aucune ressource virtuelle. Si un service ne peut être affecté aux ressources virtuelles déployées, une nouvelle ressource virtuelle est créée avec une configuration conforme aux exigences de service à satisfaire.

Etant donné que le partage des ressources influence les performances des applications d'une façon inattendue, les auteurs dans **[27]** ont proposé deux mécanismes dans le but de minimiser l'arrêt brusque des demandes suite aux dégradations de la qualité de service durant l'exécution des applications :

(i) Des contrats de performance qui établissent des relations entre les besoins de performance d'application et les capacités des ressources. Durant l'exécution, le système vérifie les données de performance pour voir si les performances voulues sont toujours respectées.

(ii) Un contrôle d'adaptation en temps réel qui permet, quand l'accord de niveau de qualité de service n'est plus respecté, d'adapter l'application aux ressources disponibles ou à la réallouer à d'autres ressources pour pouvoir satisfaire le contrat avec des performances originales.

Dans le contexte de la gestion d'adaptation de qualité de service, les auteurs dans **[5]** ont présenté un gestionnaire de ressources adaptatif en temps réel « Real-Time Adaptive Ressource Manager » qui a pour mission d'adapter la qualité de service en temps réel suivant les événements qui se produisent. Une des caractéristiques principales qui entre dans le contexte du travail de ce mémoire est que le gestionnaire alloue juste la quantité nécessaire de ressources qui permet de garder les performances demandées, même si le contrat de niveau de service mentionne plus de ressources. Ceci permet de rendre plus de ressources disponibles, et par la suite, de réduire le taux de blocage.

Nous avons traité d'une façon détaillée les travaux précédents, qui sont les plus pertinents pour notre étude. Dans le reste de cette section, nous allons brièvement parcourir le reste des travaux. Les auteurs dans **[17]** ont proposé une approche d'adaptation de QdS qui permet un recouvrement automatique, en cas de violation de QdeS : (i) par identification d'une nouvelle configuration ; (ii) par redistribution de niveau de QdS qui pourrait être supporté dans le futur; (iii) par redistribution de niveau de QdS qui pourrait être supporté immédiatement. Les auteurs dans **[28]** ont développé un système de réglage dynamique de performance, qui permet de refléter l'évolution des besoins et d'appliquer une adaptation en temps réel suivant la disponibilité des ressources. Dans **[21]**, un cadre de contrôle de QdS est proposé, permettant une décision d'adaptation dynamique de QdS et une reconfiguration des paramètres internes et des fonctionnalités d'une application multimédia distribuée. Les auteurs dans **[22]** ont présenté et examiné un modèle de négociation de QdS qui supporte la co-allocation entre différents fournisseurs de ressources. Dans **[30]**, les auteurs ont présenté un mécanisme qui optimise l'utilisation des ressources et les contraintes de QdS avec la génération des solutions sans compétitions. L'article **[4]** présente un projet nommé VIOLA qui combine la réservation des ressources de calcul, de stockage et de réseau, d'une façon qu'elle assure la disponibilité des ressources aux moments voulus. Les auteurs dans **[6]** discutent comment satisfaire le service de bout en bout avec la QdS demandée de sorte que le coût soit minimal en respectant les opérations autonomes de chaque composant du réseau ; il propose également un modèle de l'accord de niveau de service. Dans **[26]**, les auteurs proposent une approche qui permet de diviser les processus en tâches séparées, puis de les affecter d'une façon équilibrée aux différents nœuds de la grille, au lieu d'équilibrer la charge par l'application du processus de migration. L'équilibrage de la charge dans ce contexte aide à avoir un temps d'exécution court; cependant, l'extra utilisation des ressources dans une période spécifique pourrait augmenter le nombre de rejets pendant cette période.

2.3 Analyses et conclusion

Nous avons parcouru dans ce chapitre les travaux qui ont abordé les problèmes d'allocation de ressources dans une grille de calcul. L'étude de ces travaux a montré que tous les auteurs ont considéré une valeur de qualité de service pour chaque type de ressources. Ceci a pour conséquence l'augmentation du taux de blocage du fait qu'il n'y a qu'un seul choix à faire. De ce fait, nous avons proposé dans notre première contribution une approche qui donne aux utilisateurs la possibilité de spécifier plus d'une qualité de service pour un type de ressources. En plus, la plupart d'entre eux ne considèrent pas l'optimisation de l'utilisation des ressources, mais se limitent juste à l'utilisation d'un contrôle d'admission classique. Ceci a un impact également sur l'augmentation du taux de blocage et par la suite sur les pertes de revenus aux fournisseurs des services. Pour cette raison, nous avons formulé le problème comme un problème d'optimisation en utilisant un modèle de programmation en nombres entiers qui a pour fonction objectif dans notre première contribution [7] de maximiser le revenu des fournisseurs.

En ce qui concerne notre deuxième contribution **[8]**, elle se distingue des travaux faits dans le cadre d'affectation des ressources hétérogènes, par la considération de ressources non seulement hétérogènes, mais également dépendantes. Cette dépendance vient du fait des dépendances des processus, de la même requête, entre eux (à travers les communications qui existent entre eux). Ceci signifie qu'il est inutile de réserver des CPUs (par exemple, dans deux machines distribuées géographiquement) pour les processus s'il n'y a pas assez de bande passante disponible entre ces deux machines pour supporter la communication entre ces deux processus. Pour une allocation optimale de deux types de ressources dépendantes, nous avons proposé un modèle d'optimisation.

Le tableau ci-dessous résume l'essentiel des travaux antérieurs, ainsi que nos contributions.

Articles	Contrôle d'admission utilisé		Méthode de résolution utilisée			Ressources considérées		Nombre de QdS acceptables spécifiées par le client pour une ressource
	Un modèle d'optimisation	Un modèle de contrôle d'admission	Heuristique	Méthode de résolution exacte	Méthode adaptive	Type de ressources	Ressources hétérogènes dépendantes	
[1]	Oui	Non	Non	Non	Oui	Un	Non	Une
[2]	Non	Oui	-	-	-	Un	Non	Une
[3]	Non	Oui	-	-	-	Un	Non	Une
[5]	Non	Oui	Non	Non	Oui	Un	Non	Une
[7]	Oui	Non	Oui	Non	Non	Un	Non	Plusieurs
[8]	Oui	Non	Oui	Non	Non	Plusieurs	Oui	Une
[11]	Non	Oui	-	-	-	Un	Non	Une
[12]	Oui	Non	Oui	Non	Non	Plusieurs	Non	Une
[19]	Non	Oui	Non	Non	Non	Un	Non	Une
[20]	Non	Oui	-	-	-	Un	Non	Une
[23]	Oui	Non	Oui	Non	Non	Un	Non	Une
[24]	Oui	Non	Oui	Non	Non	Plusieurs	Non	Une
[25]	Oui	Non	Non	Oui	Non	Plusieurs	Non	Une
[27]	Non	Oui	Non	Non	Oui	Un	Non	Une

Tableau 1. Tableau récapitulatif

Chapitre 3

Provisionnement des ressources aux applications et services de la grilles de calcul

Mise en contexte

Dans ce chapitre, nous allons traiter le problème de provisionnement des ressources aux applications et services de la grille de calcul avec la considération d'un type de ressources (par exemple, CPU et LSP). Nous proposons pour cela un modèle de spécification des besoins de ressources qui donne la possibilité aux utilisateurs de spécifier plus d'une qualité de service pour une ressource. L'utilisateur est considéré satisfait si le minimum de qualité de service lui a été attribué, avec l'espoir d'avoir plus de qualité si les ressources deviennent disponibles avec le temps. Dans ce contexte et dans le but de maximiser les revenus des fournisseurs et de minimiser le taux de blocage, nous avons formulé le problème comme un problème d'optimisation en utilisant la programmation en nombres entiers, et nous avons également proposé deux méthodes de résolution : (i) la première nommée LOH, permet de produire une bonne solution initiale en un temps acceptable; et (ii) la deuxième nommée GOH, se lance périodiquement en arrière-plan et permet l'amélioration de la solution courante.

La première méthode (LOH) applique Cplex (outil de résolution) sur un sous ensemble de ressources, estimé avoir plus de ressources disponibles pour supporter la nouvelle demande; elle donne une solution optimale dans ce sous ensemble de ressources qui représente une solution optimale approchée dans l'ensemble des ressources. La deuxième méthode, GOH, a pour but d'améliorer en arrière-plan la solution donnée par LOH, basée sur la méthode de chaîne d'éjection entre les ressources.

Nous évaluons notre approche à l'aide des comparaisons avec d'autres techniques en termes des revenus, des taux de rejets et des temps de réponses. Le travail et les résultats présentés dans cette section ont fait l'objet d'un article intitulé « Adaptive Resources Provisioning For Grid Applications And Services» qui a été accepté dans la conférence IEEE ICC, 2008.

Adaptive Resources Provisioning for Grid Applications and Services

A. Filali and A. S. Hafid
Network Research Lab
University of Montreal, Canada
Email [redacted]

M. Gendreau
CIRRELT
University of Montreal, Canada
Email: [redacted]

Abstract

Applications utilizing Grid computing infrastructure usually require resources allocation (e.g., bandwidth and CPU) to satisfy their Quality of Service (QoS) requirements. Given the dynamic nature of grid computing, QoS support and adaptation must be a high priority to successfully support those applications. In this paper, we present an adaptive resources provisioning scheme that optimizes the resources utilization while satisfying the required QoS. More specifically, it minimizes the request blocking probability and, thus, maximizes the revenues of the infrastructure provider.

Keywords- Resources Provisioning, Adaptation, Grid, Quality of Service, Integer programming, Heuristics

I. INTRODUCTION

Grid computing provides a global-scale distributed computing infrastructure for

executing scientific and business applications [11]. Many of these applications, which have soft-real time constraints, require QoS support and assurance to execute properly. This makes it imperative to provide more stringent QoS assurances beyond those provided by the basic Grid infrastructure. In this context, a service level agreement is necessary, that specifies exactly all expectations and obligations in the business relationship between the provider and the customer [3, 4, 7]. The provider has to allocate the amount of resources, e.g., CPU and bandwidth, necessary to satisfy the agreed upon level of service. In [2], the authors present a performance tuning system that reflects changing requirements, applying real-time adaptive control technique to dynamically adapt to changing application resource demands and system resources availability. The authors in [8] present a mechanism supporting open reservations to deal with the dynamic Grid and providing a practical solution for agreement enforcement

Considerable research efforts have been dedicated to QoS management in Grid systems with particular emphasis on network resources [1,10,5,13]. Most of the existing approaches, if not all, assume one QoS value for each type of resources, e.g., 2 Mbps for bandwidth and 100 CPU cycles, during the time interval [startTime, stopTime]. The authors in [1] propose a "Grid QoS Management" framework that includes activities to manage QoS, such as enabling users to specify their QoS, selection and allocation of resources according to QoS requirements, monitoring to keep track of resources availability, etc. Particularly, they propose an adaptation algorithm that reserves resources for three types of services ('guaranteed', 'best effort', and 'adaptive'); the algorithm tries to satisfy each new request by adjusting resources reservations between the three types of services (e.g., reducing the amount of resources reserved for a "best service" request to accommodate a "guaranteed" service request). The adjustment helps avoiding the underutilisation of the Grid resources. However, the proposed approach handles only one QoS value for each type of resource.

We believe that the utilization of resources can be considerably improved by

allowing users to specify more than one value for each type of resources; indeed, this is suitable for several Grid applications for which the requirements can be satisfied using more than one value of QoS for a given resource. For example, at time currentTime, a Grid application requires the transfer of a file F (size=60 Gb) from A to B within 10 minutes (e.g., at currentTime+10, the application will have the necessary CPU cycles to process the file). This request can be satisfied using different bandwidth reservations: 1Gbps for 1 minute, 500 Mbps for 2 minutes, 250 Mbps for 4 minutes, etc. The reservation that will be selected will depend on the amount of resources available at currentTime.

In this paper, we propose an adaptive scheme that maximizes network utilisation, minimizes request blocking probability which represents a crucial factor on the user side, and thus maximizes the provider's revenues. The basic idea behind our proposal is to adjust reservations, upon receipt of a new request, upon departing an existing request or upon service degradation, in a way that maximizes the amount of reserved resources and minimize the number of requests rejected due to resources shortage. Our approach assumes that (a) a request includes a set of acceptable QoS values; and (b) one type of resources (e.g., bandwidth or CPU). We are currently working on extending the proposed approach to allow for two or more types of resources that are interrelated (e.g., bandwidth and CPU). It will be the subject of a future paper submission.

More specifically, we present an optimization problem formulation that optimizes the resource utilization/provider revenues and analyze its performance compared with a classical approach. The essence of our approach is to model resource allocation as an integer-programming problem and develop heuristics to solve, with an acceptable response time, the resulting optimization problem.

The remainder of the paper is organized as follows. Section 2 presents the optimization problem formulation. Section 3 describes a first heuristic, called LOH (Local Optimization Heuristic), for the problem resolution. Section 4 presents a second heuristic,

called GOH (Global Optimization Heuristic), for the problem resolution. Section 5 presents simulation results. Finally, Section 6 concludes the paper.

II. OPTIMIZATION PROBLEM FORMULATION

Integer programming is a technique for solving certain kinds of problems: maximizing the value of an objective function subject to constraints, where the objective function and constraints are all linear expressions. In the following, we describe a model based on binary integer programming technique and discuss how this model can be used to overcome the grid specific challenges, discussed in the introduction, in solving the resource reservation problem.

We formulate the proposed model as binary integer programming (IP) problem. Three inputs are required: a set of binary variables, an objective function, and a set of constraints (both the objective function and the constraints must be linear). IP attempts to maximize or minimize the value of the objective function by adjusting the values of the variables while enforcing the constraints. The resolution of the IP model consists of the optimal value of the objective function and the final values of the variables. In the following, we present how the proposed adjustment of resources reservation is mapped to an IP problem.

Let us define the following data and variables:

- n: represents the number of resources of the same type (e.g., a resource can be a CPU/server or an LSP in a MPLS network).
- m: represents the number of clients/requests currently being served in the system.
- E_i: represents the number of acceptable quality values for a request i (e.g., {10 Mbps, 5 Mbps, 1 Mbps}; in this case E_i=3).
- r_{ijk}: represents the portion of resource j used to satisfy the k^{th} acceptable quality value of request i.

- $R_{j,\max}$: represents the maximum capacity of resource j.
- c_{ijk} : represents unit cost when resource j is used to satisfy the k^{th} acceptable quality value of request i.
- x_{ijk} : represents a binary variable; it assumes 1 when the amount r_{ijk} of resource j is affected to request i; otherwise, it assumes 0.
- te_i : represents the end time of the request i.
- ts_i : represents the start time of the request i; te_i - ts_i represents the duration of the request.

By means of these variables, the model can be formulated as the following integer program.

Objective function

$$\text{Max} \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{Ei} c_{ijk} r_{ijk} x_{ijk} (te_i - ts_i) \qquad (1)$$

Constraints

$$\sum_{j=1}^{n}\sum_{k=1}^{E_i} x_{ijk} = 1 \quad \text{For } i = 1 \cdots m \text{ (client)} \qquad (2)$$

$$\sum_{i=1}^{m}\sum_{k=1}^{E_i} r_{ijk} x_{ijk} \le R_{j,\max} \quad \text{For } j = 1 \cdots n \text{ (resource)} \qquad (3)$$

$$x_{ijk} \in \{0,1\} \qquad (4)$$

The objective function (1) represents the provider's revenues to be maximized. Constraint (2) ensures that a request will be supported by one resource among n existing resources, constraint (3) ensures that the total amount of resources allocated does not exceed the maximum capacity of each resource, and constraint (4) ensures that the variables are binary.

III. PROBLEM RESOLUTION

The goal is to have an optimal solution (i.e., optimal adjustment of resources reservation) any time changes occur in the system. More specifically, a new user request, an existing request termination, or service degradation requires the resolution of the integer program (see Section 2). The exact optimal solution of the problem can be easily computed using any Operational Research tool (e.g., Cplex [12]). However, this will incur an unacceptable response time (e.g., in terms of hours and days), especially for large number of requests and resources.

The response time is of critical concern. Indeed, the resolution process should last a very short period of time (e.g., less than a second) to be useful. Otherwise, the system will not be able to provide a response (e.g., in response to a user request) in an acceptable time. In this paper, we define two heuristics to solve the integer program in a very short period of time compared to the exact solution. In the following, we describe the first proposed heuristic, called LOH (Local Optimization Heuristic), behaviour for each event that triggers the resolution of the program. LOH is considered as a local heuristic since it is uses Cplex on a selected subset of resources. The second heuristic, called GOH (Global Optimization Heuristic), is presented in Section 4.

A- LOH: New request

In this case, the system receives a user request *Di* that includes a list, (*Q1, Q2,...,Qm*), of acceptable quality values where *Q1* represents the minimum acceptable quality and *Qm* the most desirable quality. The system maps the list of qualities to a list of resources (*RQ1, RQ2, ..., RQm*) that are needed to satisfy the requested qualities (e.g., to satisfy *Q1*, an amount of resources equal to *RQ1* is needed). When dealing with a quality, such as bandwidth and CPU cycles, the case of most Grid applications, the mapping is straightforward since the quality and the corresponding amount of resources are the same.

The proposed heuristic starts by determining the resource j (e.g., computer j or LSP j) that has the most available resources $R_{avail,\max}$ (e.g., the most available CPU cycles or the most available bandwidth) among the n resources (e.g., n computers or n LSPs) under consideration. Then, it checks whether $R_{avail,\max}$ is bigger than $RQ1$ (i.e., $R_{avail,\max} \geq RQ1$). If the response is yes, then the heuristic determines $RQ_j \leq R_{avail,\max}$ $(1 \leq j \leq m)$, such that $j = m$ or $RQ_{J+1} \geq R_{avail,\max}$, and then allocates RQ_j to satisfy Q_j of D_i.

If $R_{avail,\max}$ is smaller than $RQ1$, the heuristic determines resource j, such that $R_{avail\,Red,\max} = Rj_{avail} + Rj_{red} = \max(Ri_{avail} + Ri_{red})$ for $1 \leq i \leq n$ where Ri_{avail} is the amount of available resources for resource i and Ri_{red} is the total amount of resources that can be reduced from requests currently being served. Let us note that the amount of resources R allocated to request k being served by resource i can be reduced by an amount equal to the difference between R and the amount of resources required to satisfy the minimum acceptable quality of request k; the amount of reduction is equal to zero if request k is minimally accommodated by the system (i.e., the system reserved just enough resources to support the minimum acceptable quality of request k). Then, the heuristic checks whether $R_{avail\,Red,\max}$ is bigger than $RQ1$ (i.e., $R_{avail,\max} \geq RQ1$). If the response is yes, then the heuristic selects a number of requests (being served by resource j) for which the amount of allocated resources is reduced (to satisfy a lower but still acceptable quality) in a way that the amount of available resources for resource j together with the sum of reductions can accommodate $RQ1$. In this case, the reservations of a number of active requests are updated with the new values (after reduction).

If $R_{avail\,Red,\max}$ is smaller than $RQ1$, then the request cannot be satisfied by a single resource. In this case, the heuristic determines resource m, such that $Rm_{avail} + Rm_{red} = \max(Ri_{avail} + Ri_{red})$ for $1 \leq i \neq j \leq n$. Then it checks whether

$(Rj_{avail} + Rj_{red}) + (Rm_{avail} + Rm_{red})$ is bigger than $RQ1$; if the response is yes, a number of requests (being served by resources j and m) for which the amount of allocated resources is reduced (to satisfy a lower but still acceptable quality) in a way to accommodate RQ1. Otherwise, the same process is repeated to consider other resources. The process terminates when RQ1 can be accommodated or when all n resources are considered without success.

B- LOH: request termination

A request termination (i.e., session corresponding to the request terminates) leads to resources being released. Thus, it is an opportunity to provide better quality to requests currently being served if not already getting the most desirable quality (included in the list of acceptable qualities of the request)

The heuristic determines, by using Uniform Random Number Generator, requests currently being served by resource j (the terminating request has been served by resource j) and increases their quality, in accordance to the list of acceptable qualities for each request, using the amount of released resources by the terminating request. With this operation, the resources utilisation is optimized and thus the provider's revenues.

C- LOH: quality degradation

Quality degradation occurs generally in case of failure; indeed, a partial or full failure of a resource will degrade the quality of requests currently being served by this resource. In the worst case scenario, the service provided to these requests is terminated (e.g., computer failure or LSP failure) and the users are notified. The proposed heuristic operates in case of quality degradation in the same way as in the case of a new request. Indeed, all failed requests are processed as new requests (see Section 3.A). One of the key challenges is to minimise service disruption; this can be achieved if the response time to failure(s) is minimized.

IV. GLOBAL OPTIMIZATION HEURISTIC

The heuristic presented in Section 3 enables local adjustment of resources in order to accommodate a new request, respond to quality degradation, or respond to request termination. We believe that these adjustments provide "local" optimization and further global optimization is possible. Indeed, if all resources and all requests being served are considered, a global optimization is needed.

In this section, we propose a second heuristic, called Global Optimization Heuristic (GOH), which can be used in conjunction with the first heuristic (LOH). The first heuristic can be used to produce an initial solution that can be optimized, periodically for example, running GOH in the background.

GOH is based on an ejection chain neighbourhood applied to GAP (generalized assignment problem) [6, 9]. It consists of moving more than one task from the current agent to a new agent. In this paper, an agent represents a resource, like a computer or an LSP, and a task represents the amount of resources required to satisfy an acceptable quality. The neighbourhood structure based on ejection chains (NSEC) was initially introduced by Glover [6] and has been applied to several problems since then. NSEC consists of two phases. In the first phase, NSEC removes a task i from an agent j, then assigns task i to a different agent w ($w \neq j$). If agent w cannot accommodate task i, NSEC, in the second phase removes a task k from agent w, assigns it to another agent z, with $z \neq w$ but it may be equal to j, and then tries to assign task i to agent w.

With some changes to NSEC, we define our heuristic GOH as follows. First, let us define S^* and L.

$$\text{Let } S^* = (X_1(x_{1.1.1}, x_{1.1.2}, \ldots, x_{1.n.e}), X_2(x_{2.1.1}, x_{2.1.2}, \ldots, x_{2.n.e}), \ldots, X_m(x_{m.1.1}, x_{m.1.2}, \ldots, x_{m.n.e}))$$

be the initial solution, given by LOH, where $X_i(x_{i.1.1}, x_{i.1.2}, \ldots, x_{i.n.e})$ represents a vector of

n*e binary variables associated with request $D_i(q_1, q_2, q_k, \ldots, q_e)$; $x_{i.j.k} = 1$ if q_k, among e acceptable qualities, is accommodated using resource j; otherwise, $x_{i.j.k} = 0$. Only a single variable assumes 1 while all others are equal to 0 (see details in Section 3).

Let *'L'* represents the list of requests for which a quality improvement is possible. A request $D_i(q_1, q_2, q_k, \ldots, q_e)$ belongs to L if and only if the amount of resources reserved for D_i supports q_j with $j \neq e$ (assuming that q_e is the most desirable quality and q_1 is the minimum acceptable quality).

The operation of GOH can be summarized as follows. GOH selects, randomly (or traverses the list L starting from the first element), a request D_h that belongs to L; D_h is associated with $X_h(x_{h.1.1}, x_{h.1.2}, \ldots, x_{h.n.e})$ that belongs to S*. If $x_{h.n.e} \neq 1$ then the following steps are executed.

Let Rq_k (required to support q_k) represents the amount of resources in resource j reserved to D_h. GOH determines resource m, among n resources, with the most available resources R_{avail}.

If $R_{avail} \geq Rq_{k+1}$, then GOH selects resource m to support D_h with the best quality possible q_l; in this case, $Rq_l \leq R_{avail}$ and ($Rq_{l+1} > R_{avail}$ or $l = e$) in the worst case scenario $l = k + 1$.

Otherwise, GOH determines a request D that satisfies the two conditions: (1) the amount of resources, Rq, currently reserved for D, in resource z, is smaller than Rq_k; and (2) Rq plus available resources in z is greater Rq_{k+1}. If D exists then, GOH reserves resources in resource z to accommodate the best quality possible for D_h (the minimum is

q_{k+1}) and reserves resources in resource j to accommodate q for D; in this case, GOH enables better utilisation of resources by providing better quality for D_h.

The process is repeated until all requests in L are considered.

V. SIMULATIONS RESULTS

In this section, we evaluate and compare a number of schemes via simulations. More specifically, we define four schemes: (1) Classical scheme: It enables the support of the best quality available to a new request; however, quality remains unchanged during the session (i.e., duration of the request). If the minimum quality cannot be supported, the request is rejected. This scheme corresponds to the behaviour of existing schemes; (2) LOH; (3) LOH+GOH: It represents a combination of LOH and GOH; LOH is used to give the initial solution while GOH is launched periodically in the background to perform global optimization; and (4) Exact solution of the IP problem.

In sub-section A, we compare the classical scheme, LOH, and LOH+GOH in terms of (averages of) revenues and rejection ratios produced by different simulations. In sub-section B, we evaluate LOH+GOH while in sub-section C we compare LOH+GOH against an exact solution (of the IP model described in Section 2).

We implemented the schemes using C++; we used CPLEX, running on Linux Machine, to resolve LOH and determine an exact solution. Table 1 shows the values of the simulation parameters. We use a simple price function: $prix(x) = 5x$, where x is the quality. More sophisticated price functions can be used.

Table 1. Simulations parameters

Simulations parameter	**Value**
Number of resources:	1000
Capacity of each resource	15

Number of requests	6000
Number of acceptable qualities	4
Quality	Generated using Uniform Random Number Generator in the interval [1,9].
Request inter-arrival time (in seconds)	Generated using Uniform Random Number Generator in the interval [0,2].
Request duration (in minutes)	Generated using Uniform Random Number Generator in the interval [15,60].
GOH Optimization periodicity	90 seconds

A- REVENUES AND REJECTION RATIO

Figure 1 shows clearly that LOH+GOH generated the most revenues when 6000 requests were served. Classic scheme generated, in the case of 6000 requests, a revenue which is almost 4 times smaller than the revenues generated by LOH+GOH; more specifically after 6000 requests have been served, the classical scheme generated 5.800E+08 while LOH+GOH generated 1.94E+09. LOH generated revenues better than classic scheme but two times smaller than LOH+GOH.

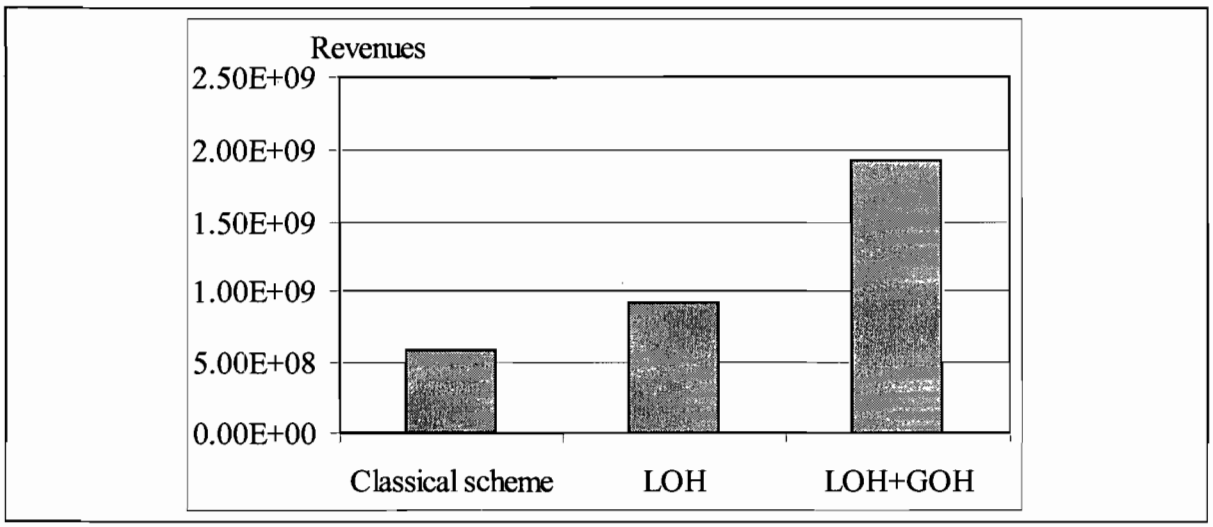

Figure1. Revenues Vs schemes

Figure 2 shows that an average of 50% of the requests is rejected when using classical scheme; this is caused by the fact that this scheme allocates the maximum requested quality, whenever possible, without any reduction of quality during the session. Using LOH the rejection ratio is about 6%; this is explained by the fact that LOH uses CPLEX on a subset of resources. CPLEX tries to exploit, in order to accept a new request, the availability in a set of resources (in opposition to a single resource) and possibly reduces the qualities of requests being served; this definitively increases the chance of accept a new request.

While LOH+GOH supports similar rejection ratio compared to LOH, it outperforms it in terms of revenues. This can be explained by the fact that LOH+GOH uses GOH in background and this increases considerably the use of resources and thus the revenues.

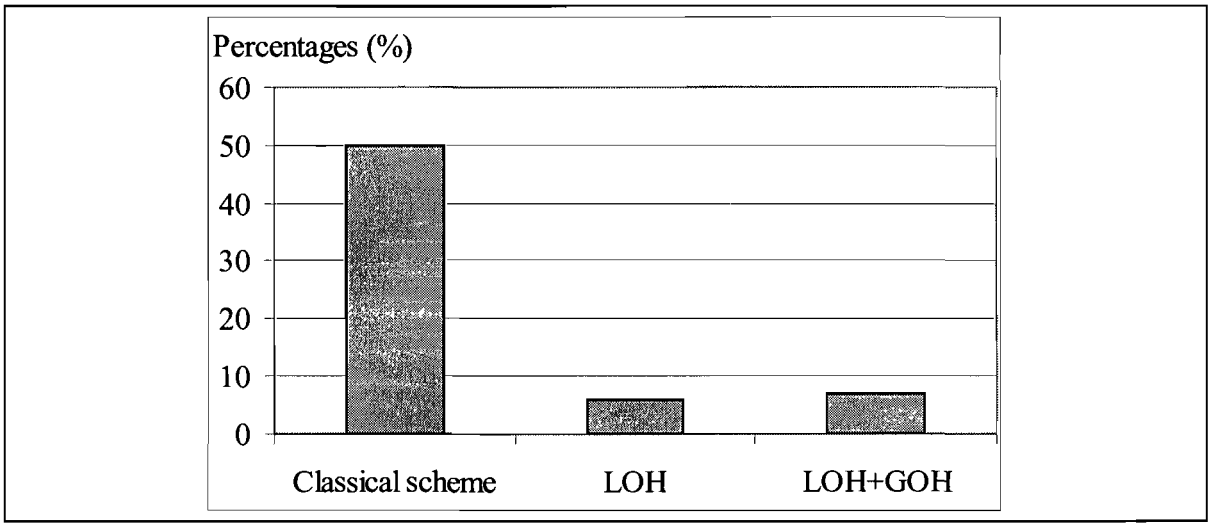

Figure 2. Rejection ratio Vs. Schemes

B- LOH+GOH: REVENUES VS. PERIODICITY

The purpose of this section is to see how the generated by LOH+GOH evolve when the periodicity of executing GOH in the background changes.

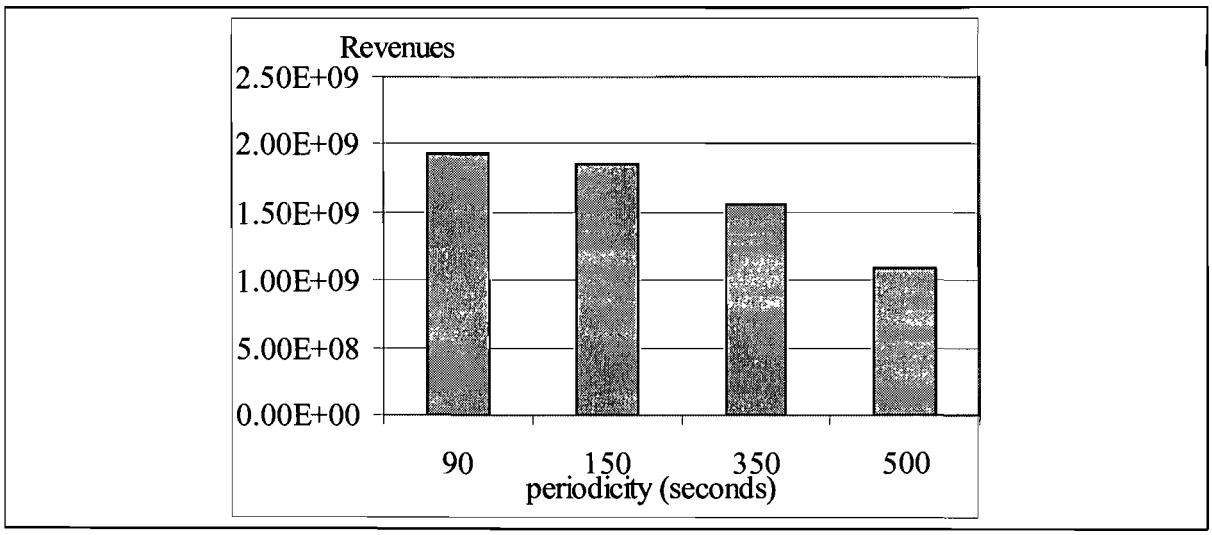

Figure 3. Revenues Vs. Periodicity

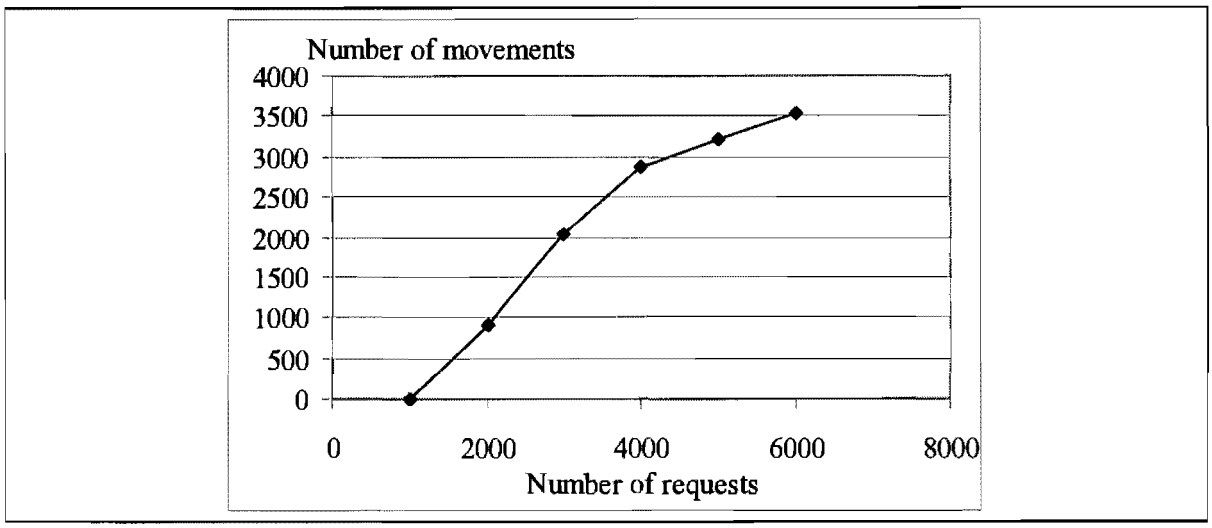

Figure 4. Movements Vs requests (using LOH+GOH)

Figure 3 shows that using smaller periodicity for running GOH in the background gives higher revenues. However, running GOH many times in the background may cause requests be moved many times among resources; this may be not acceptable in terms of the cost of moving requests from one machine to another machine or rerouting traffic from one LSP to another LSP. This being said, our simulations results indicate that the number of movements (between resources) per request does not exceed one movement for the duration of the request; Figure 4 shows that just over 50% of each 1000 requests are moved from one resource to another resource.

C- LOH+GOH Vs. EXACT SOLUTION

In this section, we compare LOH+GOH against an exact solution of the IP problem (see Section 2) in terms of revenues, number of movements per request, and response time. In general, getting an exact solution with realistic parameters (i.e., large size problem) is almost impossible, and in order to make this comparison successfully we have changed the values of the following simulations parameters: (1) Number of resources = 100 (instead of

1000); and (2) Number of requests = 170 (instead of 6000). We used Cplex for the exact solution.

The experimentations have shown that the exact solution generates slightly more revenues than LOH+GOH. The difference in terms of percentages is between 5% and 10%.

Figure 5 shows the average of movements per each request, by the end of the simulations, executed by Cplex to produce an exact solution. The average of movements decreases when the number of requests increases. This can be explained by the fact that, for each Cplex execution processing a new request, already accommodated requests are moved trying to accommodate the new request. For example, request 7 has been moved 135 times by the end of the simulations, which represents, after processing request number 150, 135 movements in 143 (150-7) executions of Cplex; it corresponds to almost one movement per Cplex execution (135/143). This is not acceptable for most applications. With LOH+GOH (Figure 4), a request is moved, on average, one time by the end of the simulations.

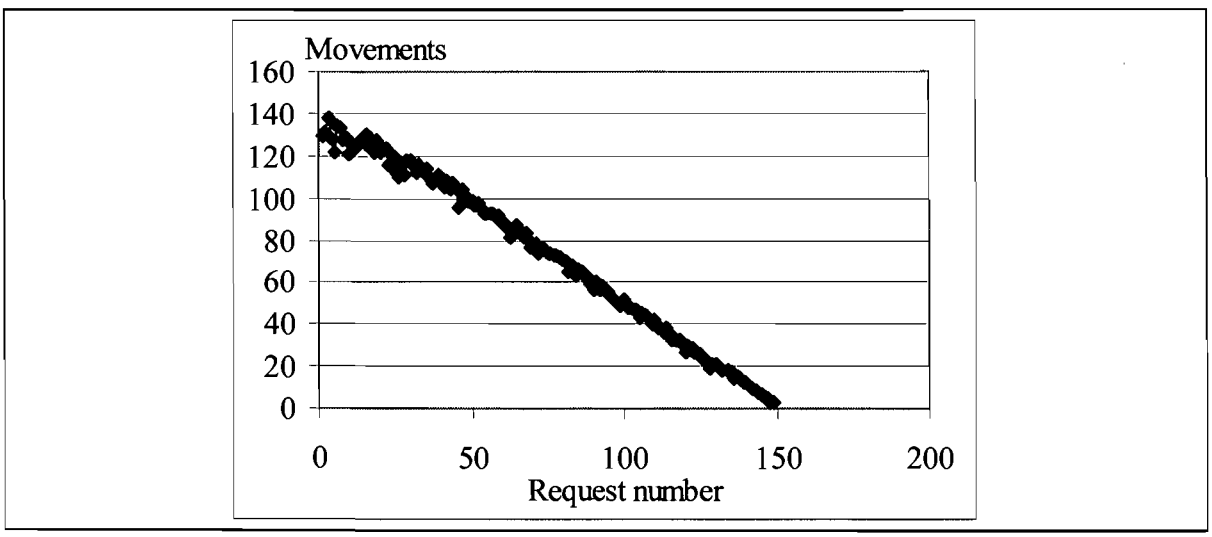

Figure 5. Exact solution: Requests Vs. movements

Figure 6 shows that the response time varies from 0 to 270 seconds (4min 30seconds) when executing Cplex to determine an exact solution while it is less than one

second when using LOH. When the number of requests exceeds 170, Cplex was not able to return an exact solution after several hours of execution (not shown in Figure 6).

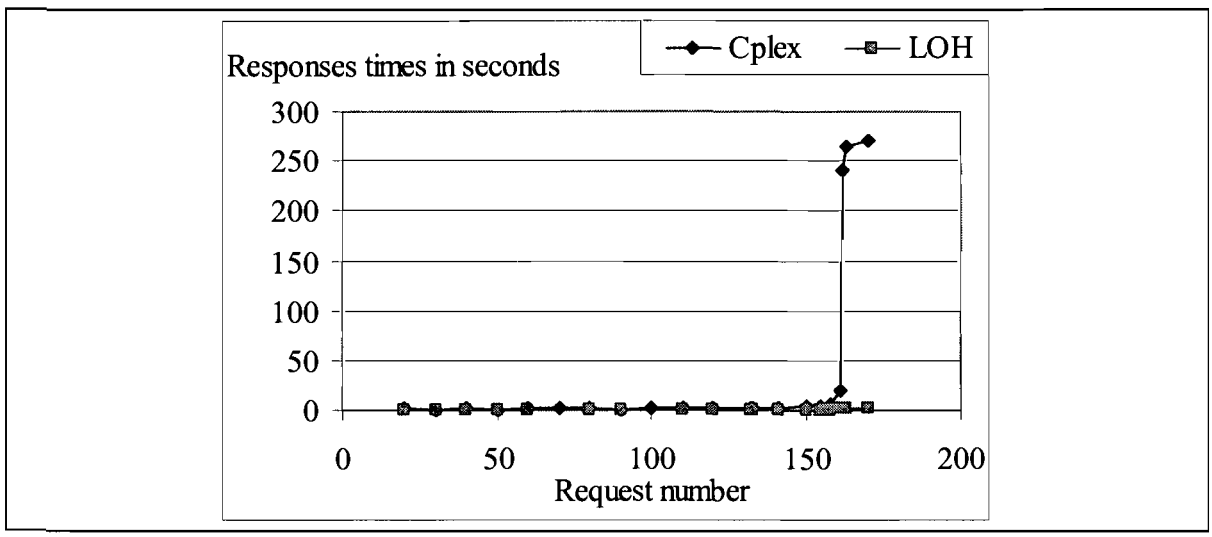

Figure 6: Responses time Vs Requests

D- ANALYSIS

We conclude that LOH+GOH is suitable to resolve the adaptive provisioning issues in Grid applications/services or any other distributed applications that require QoS assurances. In the following, we summarize our findings related to the proposed solution.

- The response time is smaller than 1 second; computing an exact solution causes a response time to exceed 5 minutes for small size problems (see Figure 6); for large size problems, the response time is "infinite".
- At a maximum a single movement, per request, between resources is required. Indeed, just over 50% of requests are moved once between resources (see Figure 4). Computing an exact solution causes several movements, per request, between resources (see Figure 5). The cost of these movements is not acceptable for most applications.

- It generates slightly less revenues than the exact solution. This is a small price to pay in order to produce results with acceptable response time and fewer movements of requests between resources.
- The rejection ratio, which presents a crucial factor for the user side, is small.

VI. CONCLUSION

In this paper, we proposed an adaptive scheme that maximizes network utilisation, minimizes request blocking probability, and maximizes the provider's revenues. The basic idea behind our proposal is to adjust reservations, upon receipt of a new request, upon departing an existing request or upon service degradation, in a way to maximize the amount of reserved resources and minimize the number of requests rejected due to resources shortage.

More specifically, we developed an optimization model using BIP (binary integer programming). Then, we defined a heuristic (LOH) to resolve BIP with an acceptable (a) response time and (b) number of movements per request. A second heuristic was developed in order to be used in combination with LOH for better optimization of resources utilization. The simulation results show that LOH+GOH outperforms existing provisioning schemes in terms of revenues and rejection ratio. It also provides better performance than an exact solution while providing slightly less revenues.

Currently, we are working on extending the proposed model to consider more than one type of interrelated resources (e.g., CPU and bandwidth).

REFERENCES

[1] R. Al-Ali, A. Hafid, O. Rana and D. Walker, *An Approach for QoS Adaptation in Service-Oriented Grids, The Journal of Concurrency: Practice and Experience*, Vol. 16, Issue 5, Pages 401- 412 , 2004.

[2] B. Doshi, S. Wang, P. Kim; D. Goldsmith, B. Liebowitz, K. Park, *Cooperative Service Level Agreement, Military Communications Conference*, 2006.

[3] I. Foster and C. Kesselman, eds., *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 2003.

[4] I. Foster and A. Roy, *A Quality of Service Architecture that Combines Resources Reservation and Application Adaptation, 8th International Workshop on Quality of Service* (IWQOS 2000), 2000.

[5] F. Glover, *Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problem, Discrete Appl. Math.*, Vol 65, No.1- 3, Pages 223-253, 1996.

[6] A. Hafid and G. Bochmann, *Quality of Service Adaptation in Distributed Multimedia Applications, ACM Springer-Verlag Multimedia Systems Journal*, Vol. 6, No. 5, Pages 299- 315, 1998.

[7] *ILOG CPLEX.* http://www.ilog.com

[8] B. Li and K. Nahrstedt, *A Control-Based Middleware Framework for Quality-of-Service Adaptations, IEEE,* Vol. 17 Issue 9, Pages 1632- 1650, 1999.

[9] J. Li, R.Yahyapour, *A Negotiation Model Supporting Co-Allocation for Grid Scheduling, 7th IEEE/ACM International Conference on Grid Computing*, 2006.

[10] D. A. Reed and C. L. Mendes, *Intelligent Monitoring for Adaptation in Grid Applications, In Proceedings of the IEEE*, Vol. 93, Issue 2, Pages 426- 435, 2005.

[11] R. L. Ribler, J, S. Vetter, H Simitci, and D. A. Reed, *Autopilot: Adaptive Control of Distributed Applications, 7th IEEE Symp. On High Performance Distributed Computing*, 1998.

[12] M. Siddiqui, A. Villaz and T. Fahringer, *Grid capacity planning with negotiation-based advance reservation for optimized QoS, ACM/IEEE conference on Supercomputing*, 2006.

[13] M. Yagiura, T. Ibaraki, F. Glover, *An Ejection Chain Approach for the Generalized Assignment Problem*, Vol 16, Pages 133-151, 2004.

Chapitre 4

Provisionnement des ressources aux applications et services de la grille de calcul

Mise en contexte

Dans ce chapitre, nous allons traiter le problème de provisionnement des ressources aux applications et services de la grille de calcul avec la considération de plusieurs types de ressources hétérogènes dépendantes (par exemple, CPU et LSP). Étant donné que l'infrastructure de la grille de calcul est un ensemble de ressources distribuées géographiquement, le support des applications de grille de calcul nécessite l'affectation des ressources de calcul et de bande passante pour permettre la communication entre ces ressources. Notre objectif est d'accepter le maximum possible d'applications avec la satisfaction de leurs besoins. Idéalement, nous voudrions accommoder ces applications avec l'utilisation des ressources de la grille (par exemple, un serveur) non distribuées géographiquement (par exemple, dans le même réseau local). Cependant, cette possibilité n'est pas toujours réalisable. De ce fait, pour pouvoir augmenter la probabilité d'accommodation des applications de grille de calcul, l'utilisation des ressources éparpillées à travers le réseau peut être une éventualité, et dans ce cas, l'accommodation de bande passante est une nécessité pour la communication entre ces ressources à travers le réseau. Dans ce contexte, nous proposons un modèle d'optimisation d'affectation 'simultanée' des ressources de calcul et de bande passante pour les applications de grilles de calcul avec la maximisation du nombre d'applications accommodées. Une heuristique de résolution composée de trois phases est proposée : (i) les deux premières phases produisent une bonne solution de départ; (ii) la troisième phase, qui utilise la méthode tabou, sert à améliorer la solution de départ.

Nous évaluons notre approche à l'aide des comparaisons avec d'autres techniques en termes des temps de réponses, des taux de rejets, et des coûts d'utilisation de bande passante. Le travail et les résultats présentés dans cette section ont fait l'objet d'un article intitulé « Bandwidth and Computing Resources Provisioning for Grid Applications and Services» qui a été soumis dans la conférence IEEE GC, 2008.

Bandwidth and Computing Resources Provisioning for Grid Applications and Services.

A. Filali, A. S. Hafid
NRL, University of Montreal, Canada
Email [redacted]

M. Gendreau
CIRRELT , University of Montreal, Canada
Email: [redacted]

Abstract:

Applications using Grid computing infrastructure usually require resources allocation to satisfy their Quality of Service (QoS) requirements. Given that the Grid infrastructure is a set of computing resources geographically distributed, the support of Grid applications requires the allocation of computing resources and bandwidth to enable communication among these resources. The objective is to accommodate as many applications as possible while still satisfying their requirements. Ideally, we would like to accommodate a given Grid application using a set of computing resources (e.g., one server) that are not geographically distributed (e.g., in the same LAN); however, this is not always possible. Indeed, to increase the probability of accommodating Grid applications, we may need to use computing resources scattered all over the network; in this case, bandwidth allocation is required to enable communication among these resources. In this paper, we propose an optimization model that enables the "simultaneous" allocation of computing resources and bandwidth for Grid application while maximizing the number of Grid applications being accommodated. A heuristic is proposed to solve the model with an

acceptable response time; simulations show that the proposed approach outperforms existing classical approaches.

Keywords- *Bandwidth, Grid, Quality of Service, Integer programming, Heuristics*

I. INTRODUCTION

Grid computing provides a global-scale distributed computing infrastructure for executing scientific and business applications [5, 6]. As Grid computing becomes more popular, the size and the diversity in terms of resources and applications increase as well. In fact, many applications require the allocation of more than one type of resources to execute properly [14, 11, 3, 2].

Considerable research efforts have been dedicated to the allocation of resources to support agreed-upon QoS (Quality of Service) [1, 7]. In [15], the authors present a performance tuning system that applies real-time adaptive control techniques to dynamically adapt to changing application resource demands and system resources availability. The tuning in this context helps avoiding terminations of applications/services due to quality degradation. The authors in [16] present a mechanism supporting open reservations to deal with the dynamic Grid and to provide a practical solution to enforce agreed-upon QoS. In [13], the authors propose an approach that calls for splitting up processes into separate jobs and then balancing them to nodes instead of balancing the load in the Grid by process migration or by moving an entire process to a less loaded node. Load balancing helps shortening the execution time; however, extra resources utilization in a specific period could lead to an increase in the rejection ratio during that period.

The authors in [1] propose a Grid QoS Management framework that includes activities to manage QoS, such as QoS specification, selection and allocation of resources according to QoS requirements, monitoring to keep track of resources availability, etc. In particular, they propose an adaptation algorithm that reserves resources for three types of

services ('guaranteed', 'best effort', and 'adaptive'); the algorithm tries to satisfy each new request by adjusting resources reservations between the three types of services (e.g., reducing the amount of resources reserved for a "best service" request to accommodate a "guaranteed" service request). The adjustment helps avoiding the underutilisation of Grid resources and maximizing the provider's revenues.

The authors in [12] propose a model for the resource matching problem and solve it, as an on-line optimization problem, using mixed integer programming methods; they describe how the resource dependencies, capacity requirements and constraints can be modeled. The objective is to maximize throughput, and thus minimize the blocking probability; they use CPLEX (exact solution method) to solve the model. Using an exact method resolution could take an "infinite" response time for realistic large size problems. In [4], the authors present an adaptive resource provisioning scheme that optimizes the resources utilization while satisfying the required QoS. More specifically, it minimizes the request blocking probability and, thus, maximizes the revenues of the infrastructure provider. However, the proposed scheme can be used only for the provisioning of a single type of resources.

In this paper, we propose a provisioning scheme for Grid applications/services that minimizes the blocking probability (and thus increase the provider's revenues); it enables the joint optimal allocation of two types of dependent resources, namely computing resources and bandwidth. Ideally, we would like to accommodate a given Grid application (that consists of a set of processes) using a set of computing resources (e.g., servers) that are not geographically distributed (e.g., in the same LAN); in this case, there is no need for bandwidth allocation. However, this is not always possible. Indeed, to increase the probability of accommodating Grid applications, we may need to use computing resources scattered all over the network; in this case, bandwidth allocation is required to support communication among the application processes. To support optimal "joint" allocation of computing resources and bandwidth, we propose an optimization model (formulated as an Integer Program) that minimizes the use of bandwidth (thus, it maximizes the revenues of

the provider) and maximizes the throughput (thus, it minimizes the blocking probability) at the same time. The optimization problem being NP-hard, a heuristic is then proposed to solve it, thus providing an acceptable response time for realistic large size instances.

The remainder of the paper is organized as follows. Section 2 presents the optimization problem formulation. Section 3 describes the solution method. Section 4 presents simulations and analysis. Section 5 presents the model formulation with timing parameters. Finally, Section 6 concludes the paper.

II. OPTIMIZATION PROBLEM FORMULATION

Integer programming (IP) is a technique for optimizing the value of an objective function subject to constraints, where the objective function and constraints are functions defined over a set of decision variables that are restricted to taking integer values. In the following, we propose an IP model for tackling the Grid resource reservation problem discussed in the introduction.

Let us first define the data (inputs) and the variables required by the model before stating its objective and constraints.

Data:

1. R: the set of (provisioning) requests r.
2. P^r: the set of processes included in request r.
3. $P = \cup P^r$: the set of all processes in the system.
4. D: the set of domains s; a domain consists of a set processors that do not need WAN bandwidth for communication among themselves (e.g., processors in a LAN). Two processors belonging to different domains require bandwidth for communication.
5. M_s: the set of processors belonging to domain s.
6. $M = \cup M_s$ for $s \in D$.
7. E: the set of edges e that link adjacent domains.
8. L^{st}: the set of paths l between domains s and t.
9. $L = \cup L^{st}$ for $s, t \in D$.
10. L_e: the set of paths l to which edge e belongs.
11. w_l: the cost of using path l.

12. b_{ijr}: the bandwidth required when processes i and j of request r are assigned to different domains.
13. k^m: the capacity of processor $m \in M$.
14. c_i: the amount of computing capacity required by process i.

Variables:

1. z_{ijr}^l: binary variable that indicates whether path l is used for traffic between processes i and j of request r; $z_{ijr}^l = 1$, if path l is used, and 0 otherwise.
2. x_{ir}^m : binary variable that indicates whether process i of request r is assigned to processor m; $x_{ir}^m = 1$, if process i is assigned to m, and 0 otherwise.
3. y_{ir}^s : binary variable that indicates whether process i of request r is assigned to domain s; $y_{ir}^s = 1$, if process i is assigned to s, and 0 otherwise.
4. v_l: the amount of bandwidth carried by path l.

Objective function:

$$\text{Min} \sum_{l \in L} w_l \, v_l \qquad \textbf{(1)}$$

Constraints:

$$y_{ir}^s = \sum_{m \in M_s} x_{ir}^m, \quad s \in D, i \in P^r, \, r \in R. \qquad \textbf{(2)}$$

$$\sum_{s \in D} y_{ir}^s = 1, \quad i \in P^r, \, r \in R. \qquad \textbf{(3)}$$

$$\sum_{l \in L^{St}} z_{ijr}^l \geq y_{ir}^s + y_{jr}^t - 1, \quad i, j \in P^r, r \in R, s,t \in D. \qquad \textbf{(4)}$$

$$v_l = \sum_{r \in R} \sum_{i, j \in P^r} b_{ijr} z_{ijr}^l, \quad l \in L. \qquad \textbf{(5)}$$

$$\sum_{l \in L_e} v_l \leq k_e, \quad e \in E. \qquad \textbf{(6)}$$

$$\sum_{r \in R} \sum_{i \in P^r} c_i x_{ir}^m \leq k^m, \quad m \in M. \qquad \textbf{(7)}$$

$$z_{ijr}^l \in \{0,1\}, \quad i, j \in P^r, r \in R, l \in L. \qquad \textbf{(8.a)}$$

$$x_{ir}^{m} \in \{0,1\}, \quad i \in P^{r}, r \in R, m \in M. \qquad \textbf{(8.b)}$$

$$y_{ir}^{s} \in \{0,1\}, \quad i \in P^{r}, r \in R, s \in D. \qquad \textbf{(8.c)}$$

$$0 \leq v_{l}, \quad l \in L. \qquad \textbf{(8.d)}$$

The objective function (1) represents the cost of bandwidth utilization, which should be minimized when the requests are served. Constraint (2) ensures that each process i of request r is assigned to one processor m of domain s when it is assigned to this domain. Constraint (3) ensures that process i of request r is assigned to one domain s. Constraint (4) ensures that a path is assigned to connect domains s and t on which different processes i and j of any given request r are assigned. Constraint (5) computes the overall bandwidth requirement for paths. Constraint (6) ensures that the capacity of each edge e is not exceeded. Constraint (7) ensures that the capacity of each processor m is not exceeded. Constraints (8.a, 8.b, 8.c, 8.d) define the variables of the formulation.

III. PROBLEM RESOLUTION

The goal is to have an optimal solution whenever requests arrive. The exact optimal solution of the problem can be computed using appropriate operations research tools (e.g., CPLEX); however, this may require several hours, or even days, when the number of requests and resources is large.

Since response time is of critical concern, the solution process should not require more than a second or so. We thus propose in this paper a heuristic to solve the integer program much faster than an exact solution method.

Our heuristic is made up of three phases: (1) The first produces an initial solution that may be infeasible because some processes could not be allocated; (2) The second completes an initial solution by allocating the processes that could not be allocated in the

first phase; and (3) The third is devoted to improving the initial feasible solution produced earlier using a Tabu Search algorithm.

A- BUILDING AN INITIAL FEASIBLE SOLUTION

First phase

In order to compute a good partial initial solution quickly, we use a greedy method. More specifically, we scan the list of requests in decreasing order of CPU cycle requirements. Each request is examined in turn. For each request, processes are considered individually, again in decreasing order of CPU cycle requirements. If a process can be assigned (i.e., if there is a processor with sufficient capacity remaining and, possibly, enough remaining bandwidth to support communications with processes already assigned), the partial solution is updated; otherwise, the process is put into a list, WAITLIST, of waiting processes. At the end of this phase, we obtain a good partial initial solution S* and a list of waiting processes, which may be empty. In this case, there is no need to execute the second phase and we can proceed directly to phase 3.

Second phase

The objective of this phase is to assign, if possible, the processes in *WAITLIST*. The basic idea is to revisit existing assignments and check whether by reshuffling we can assign processes in *WAITLIST*. More specifically, we select the process *ps*, in *WAITLIST*, with the largest amount of CPU cycles and the processor *pr* with the most available CPU. Then, we move one or more processes from pr to other processors (that can accommodate them) in order to make room for ps. If this operation succeeds (i.e., one can find sufficient available CPU and sufficient bandwidth to enable the required communication between *ps* and processes already assigned), we make the corresponding CPU and bandwidth reservation, we remove *ps* from *WAITLIST*, and we consider the next process in *WAITLIST*. Otherwise, we consider a processor (different from *pr*) with the most available CPU and we repeat the

same operation. This terminates when *WAILIST* is empty (a feasible solution is found) or no feasible solution can be found. The pseudo-code is presented in Figure 1.

```
Boolean complete_initial_solution(S, WAITLIST)
Begin
 all_inserted = true;
 While (still process in WAITLIST and all_inserted == true)
   {- select the best process ps in WAITLIST
   - inserted = false;
   - j = 0;
   - while (inserted == false and j < number_of_processor)
       { - select processor pr having the most CPU cycles available
         - if (allocate_ps_in_pr(ps, pr)==true)
            {inserted=true;
             Update S*;
             Remove process ps from WAITLIST;}
          Else
            { j++;          }
        }
    if (inserted==false)
      { all_inserted=false}
  }
If (all_inserted == true)
     { /* initial solution S* is found */
       Return (true)}
Else
     { /* initial solution can not be found */
       Return (false)}
```

```
End complete_initial_solution
Boolean allocate_ps_in_pr(int ps, int pr)
Begin
{ a- check whether the requested CPU amount of process ps is smaller than the available
  CPU amount in processor pr.
  b- if yes: check if needed_bandwidth(ps)=true.
        If yes: return (true)
        Else: return (false)
  c- else: stop=false
        while (still processes in pr and stop == false)
          {if (move_one_proccess_from_pr(pr)==true)
              { stop = true }
          }
        if (stop == true) : repeat from a;
        else : return (false);
}
End allocate_ps_in_pr

Boolean move_one_proccess_from_pr(int pr)
Begin
  { a. Select a process pm, assigned to pr, to move;
    b. if found
       c. find a processor to where it can be moved /* processor with sufficient available
          CPU to support pm */
       d. if found
          e. Check whether the needed bandwidth can be reserved.
          f. If yes return (true)
          g. else find another processor and repeat from d.
```

h. else select another process '*pm*' and repeat from **b.**

i. else return (false).

}

End move_one_proccess_from_pr

Boolean needed_bandwidth(int *ps*)

Begin

{

a- determine the bandwidth required to enable communication between *ps* and its associated processes $\{p_1,\ldots,p_k\}$ (*ps*, $p_1,\ldots,p_k$ belong to the same provisioning request) that are assigned to different domains /* *ps*, $p_1,\ldots,p_k$ belong to the same provisioning request*/

b- for each pair (p_s, p_i) where $1 \prec i \leq k$, select the path, with the most available bandwidth, between the domain where p_i is assigned and the domain where *ps* is considered for assignment.

c- if for each pair (p_s, p_i), the selected path can accommodate the requested bandwidth between *ps* and p_i, then make the corresponding bandwidth reservation and return true.

d- otherwise, return false.

}

End needed_bandwidth

Figure 1. Phase 2: pseudo-code

B- IMPROVING THE INITIAL SOLUTION: Third phase

A Tabu Search heuristic is used in this phase to improve the initial solution found in step 2. First proposed by Glover [9] in 1986, Tabu Search can be seen as an extension of classical local search techniques. Its main feature is the use of memory structures that are used to overcome local optima and to guide the exploration of the solution space. Cycling is prevented by forbidding the reversal of previous moves by making these "tabu" (hence the name of the method). Information on these tabus is recorded in short-term memories called the tabu lists (see [8] for more details).

A key ingredient of any local search procedure is the definition of the modifications that can be performed on the current solution at any iteration. Since this defines which

solutions are adjacent to the current solution, it is called the neighborhood structure of the search procedure.

In our problem, since our objective is to minimize the bandwidth cost we need to move processes from one domain to another to reduce the amount of bandwidth used to communicate between processes of the requests. The neighborhood structure is defined around domains and aims at exchanging as many processes as possible between the processors of a pair of domains. For each pair of domains (s, t) of M, we try to exchange as many processes as possible between each pair of processors belonging respectively to those two domains. We repeat this procedure for all domain pairs (s, t) and select the pair (s^*, t^*) that yields the greatest improvement in the objective. The exchange of processes between the processes of s^* and t^* is then performed and leads to the new current solution. This process is repeated until a (predefined) maximum number of movements is reached. See the pseudo-code of Figure 2 for more details.

```
Optimize()
Begin
{int  mvt =1, max_mvt /*max movement*/, nb_domain /*number of domains*/;
 Int i, j;
 int i_save, j_save;/* save domains indexes that give less cost*/
 float temp_current_value = total_cost;
 bool improve = false;
 float new_cost;
 while (mvt ≤ max_mvt && improve == false)
  {for (i = 0; i ≺ nb_domain; i++;)
   {for (j = i + 1; j ≺ nb_domain; j++;)
    {new_cost= exchange(i, j, 'A'); /*audit procedure call*/
     if (new_cost ≺ total_cost ))
      {i_save = i;
```

```
        j_save = j;
        temp_current_value = new_cost;
       }
      }
    }
   if (total_cost ≻ temp_current_value)
    { total_cost = exchange(i_save, j_save, 'U');
      mvt+=1;
    }
   else
    {improve = true;
    }
  }
}
```

End optimize

float exchange(int *i*, int *j*, char *mode*) /*exchange procedure between domains *i* and *j*/

float *new_cost* = M; // M big number

Begin

{/**mode*='A' : audit mode, *mode*='U' : update mode*/

int *t, p;*

for $(t = 0; t \prec domains[i] \times nbr\_processors; t++;)$

{

a. Select processor *p* from domain j that verifies the three following conditions:
- o *Processor[p].used_capacity* <= *Processor[t].total_capacity*
- o *Processor[t].used_capacity* <= *Processor[p].total_capacity*
- o Processor *p* is not already participated in exchange

b. If *found*
- o by switching the processes between processors *t* and *p* check the reservation of the necessary bandwidth for each communication if it is possible.
- o If all reservations could be done

```
        ▪ Update new cost
        ▪ Make the updates effective if in Update mode
      o Else repeat from a.
    } // end of for
  Return (new_cost)
}
End exchange;
```

Figure 2. Third phase: pseudo-code

IV. SIMULATIONS AND ANALYSIS

In this section, we evaluate and compare a number of schemes, via simulations, in terms of (average) cost and rejection ratios for satisfying a set of provisioning requests. More specifically, we define four schemes: (1) classical scheme: allocate first the requested amount of CPU and then the requested amount of bandwidth for traffic between processes; (2) greedy scheme: determine a solution using a greedy algorithm (first phase, Section III-A); (3) GI scheme: determine a solution using a greedy algorithm combined with an insertion procedure (Section III-A); and (4) GIT scheme: combination of GI and Tabu Search (Section III-B).

The simulations are performed by varying the number of domains, the number of processors in each domain and the network topology that links the domains; we vary also the number of requests submitted in a batch, the number of processes and bandwidth required between processes inside each request. Tables 1-3 show the parameters used in our simulations. In Tables 1-2, we show the parameters of three simulation setups. Table 1 displays the details of the system under simulation; for example, the first setup consists of 4 domains, 16 processors with 70 of capacity each, 30 paths and 6 edges with 35 of capacity each. Table 2 shows, for the first setup, that 30 requests with a total of 90 processes are handled at a time, and the total number of communications between processes is 90 (in average 3 processes per request and each process, in a request, needs to communicate with the other 2 processes). Table 3 shows the range of values of CPU per process, bandwidth

between any 2 processes of the same request, and the cost of using a specific path. Specific values are generated using uniform random generator.

Table 1. Mesh setups

Setup	Domain	Prs	P_cap	Path	Edgs	E_cap
1	4	16	70	30	6	35
2	8	32	70	86	9	50
3	16	64	70	240	19	50

Table 2. Request setups

Setup	Request	Processe	Traffic
1	30	90	90
2	120	360	360
3	240	720	720

Table 3. Ranges of values

Parameter	Range of values	Distribution function
CPU (1)	[4 .. 20]	C++ uniform random function
Bandwidth (2)	[0.01 .. 2]	C++ uniform random function
Path cost (3)	[2 .. 10]	C++ uniform random function

We implemented the schemes using C++ running on Linux. For the four schemes, we run the simulations 20 times and we compute the average values observed for:

1. the response time for computing a solution,
2. the cost of the solution,
3. the rejection ratio.

We now discuss the results for each of these measures.

A- RESPONSE TIME

Figure 3 shows that all schemes give a good response time, in milliseconds. Even if GIT scheme has the longest response time, it still does not exceed 200 ms in all cases.

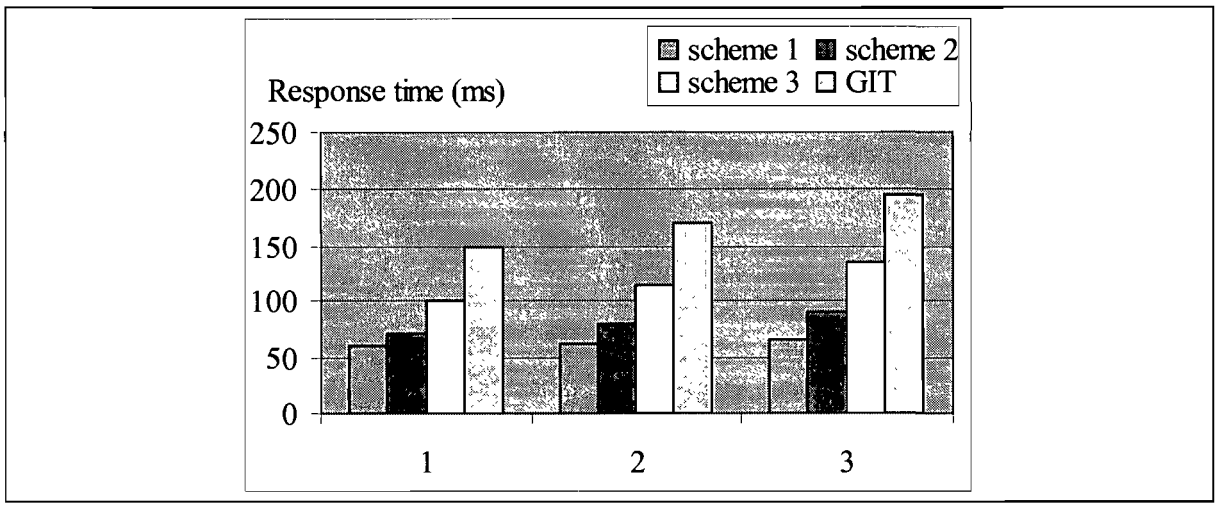

Figure 3. Average responses times (ms) for the 3 setups

B- UTILISATION COST

Figure 2 shows that GIT is the least costly scheme for accommodating the set of provisioned requests. More specifically, under GIT, the cost is 18% lower than for scheme 1, 26% lower than for scheme 3, and 16% lower than for scheme 2.

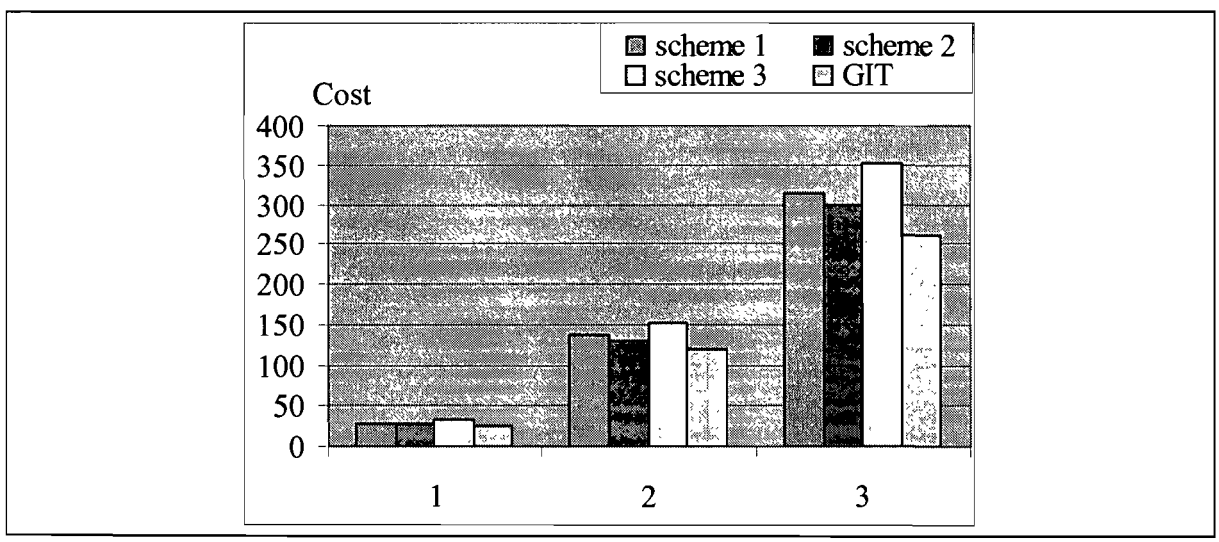

Figure 4. GIT vs. other schemes: Cost

C- REJECTION RATIOS

Figure 5 shows that on average 10% of requests are rejected when using scheme 1, 4% are rejected when using scheme 2, and only 2% when using scheme 3 or GIT.

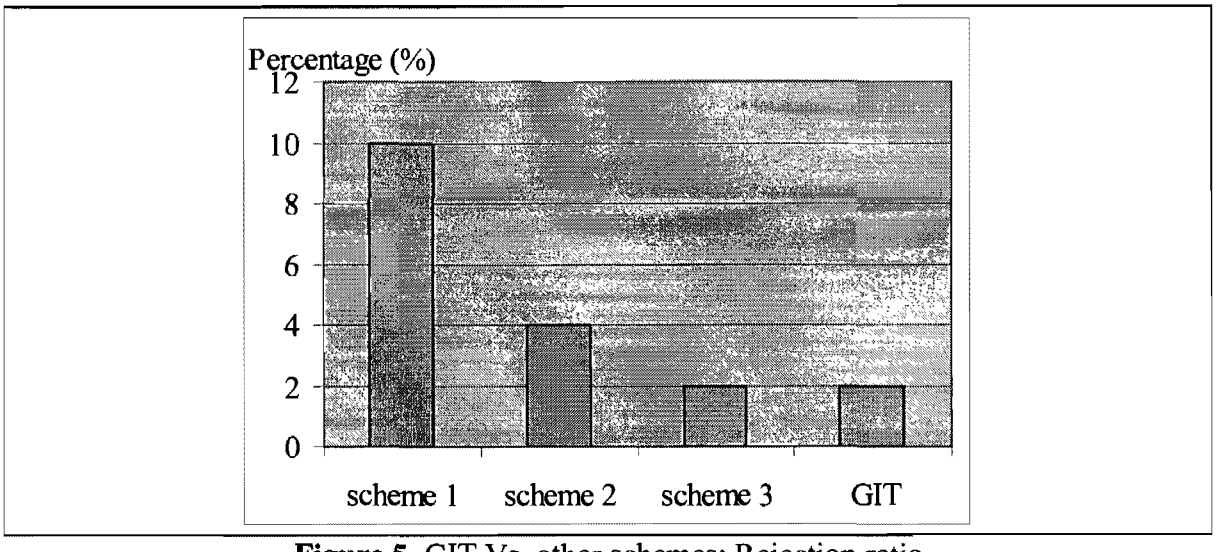

Figure 5. GIT Vs. other schemes: Rejection ratio

D- ANALYSIS

We conclude that GIT is suitable for resource provisioning in Grid applications/services and all other distributed applications that require bandwidth and CPU resources. In the following, we summarize our findings related to the proposed solution.

- The response time is smaller than 1 second (Figure 3).
- Compared to the three other schemes described in section IV, GIT produces the smallest utilization cost while satisfying the requirements of the incoming requests.
- The rejection ratio, which presents a critical factor for users, is the smallest for GIT.

V. PROBLEM WITH TIMING PARAMETERS

Our approach can be easily adapted if we take into account timing parameters: start time and end time for each request (or each process of a request). The goal is to have an

optimal solution based on the available resources at a specific time; the resources allocated for the requests already being served are not considered in the optimization. The objective function with the timing parameters can be formulated as below:

$$\text{Min} \sum_{r \in R} \left[(t_{end,r} - t_{start,r}) \sum_{l \in L} w_l v_{lr} \right]$$

Subject to the same constraints as in Section II, where

$t_{end,r}$ denotes the ending time of request r;

$t_{start,r}$ denotes the starting time of request r:

v_{lr} denotes the amount of bandwidth used by request r, on path l, at time $t_{start,r}$.

The solution heuristic GIT is as in Section 3 but with the introduction of the time parameters in the objective.

VI. CONCLUSION

In this paper, we have proposed an approach that enables the provisioning of bandwidth and CPU resources for Grid applications and services. The basic idea behind our proposal is to allocate as many processes as possible of the same request to the same domain; the goal is to minimize the use of bandwidth and eventually reduce request failures due to network failures.

More specifically, we developed an optimization model using integer programming. Then, we defined a solution heuristic (GIT) to resolve the model with an acceptable response time, minimum cost, and low rejection ratio. The simulations results show that GIT gives a good response time in millisecond and outperforms classical schemes in terms of cost and rejection ratio. With respect to the proposed tabu search algorithm, other neighbourhood structures could be easily implemented, e.g., performing the exchange at

the level of processes between two domains instead of exchange at the level of processors between domains.

REFERENCES

[1] R. Al-Ali, A. Hafid, O. Rana and D. Walker, *An Approach for QoS Adaptation in Service-Oriented Grids, The Journal of Concurrency: Practice and Experience*, Vol. 16, Issue 5, pages 401- 412, 2004.

[2] C. Barz, M. Pilz, T. Eickermann, L. Kirtchakova, O. Wäldrich and W. Ziegler, *Co-Allocation of Compute and Network Resources in the VIOLA-Testbed* 2006.

[3] B. Doshi, S. Wang, P. Kim, D. Goldsmith, B. Liebowitz, K. Park, *Cooperative Service Level Agreement, Military Communications Conference*, pages 1-7, 23-25 Oct 2006.

[4] A. Filali, A. Hafid, M. Gendreau, *Adaptive Resources Provisioning for grid applications and services, IEEE ICC'08*, China, 2008.

[5] I. Foster, C. Kesselman, S. Tuecke, *The anatomy of the Grid enabling scalable virtual organizations, International J. Supercomputer Applications*, Vol. 15, No. 3, pages 200-222, 2001.

[6] I. Foster and C. Kesselman, eds., *The Grid 2 Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers*, Los Altos, CA, 2004.

[7] I. Foster and A. Roy, *A Quality of Service Architecture that Combines Resources Reservation and Application Adaptation. In Proceedings of 8th International Workshop on Quality of Service*, Pittsburgh, PA, pages 181-188, 2000.

[8] M. Gendreau and J.-Y Potvin, *Search methodologies: Introductory Tutorials in Optimization and Decision Support Techniques (edited by Edmund K. Burke, Graham Kendall)*, Ch. 6 *"Tabu Search"*, pages 165-186, *Springer*, 2005.

[9] F. Glover *"Future Paths for Integer Programming and Lines to Artificial Intelligence", Computer and Operations Research* 13, pages 533-549, 1986.

[10] G. Gutin, A. Yeo and A. Zverovich, *Introduction to Algorithms (Cormen, Leiserson, and Rivest)*, Ch 16 "*Greedy Algorithms*", 2001.

[11] J. Li, R.Yahyapour, *A Negotiation Model Supporting Co-Allocation for Grid Scheduling, The 7th IEEE/ACM International Conference on Grid Computing*, Barcelona, Sept 28-29, IEEE Computer Society Press, 2006.

[12] V. K. Naik, C. Liu, L. Yang and J. Wagner, *Online resource matching for heterogeneous grid environments,* In Proceedings of *5th IEEE International Symposium on Cluster Computing and the Grid,* Vol. 2, pages 607-614, 2005.

[13] N. Nehra, R.B. Patel and V.K. Bhat, *Distributed Parallel Resource Co-allocation with Load Balancing in Grid Computing, IJCSNS*, Vol. 7, No.1, pages 282-292, 2007.

[14] D. A. Reed and C. L. Mendes, *Intelligent Monitoring for Adaptation in Grid Applications, In Proceedings of the IEEE*, Vol. 93, Issue 2, pages 426- 435, February 2005.

[15] R. L. Ribler, J, S. Vetter, H Simitci, and D. A. Reed, *Autopilot: Adaptive Control of Distributed Applications, In Proceeding of 7th IEEE Symp. On High Performance Distributed Computing*, 1998.

[16] M. Siddiqui, A. Villaz and T. Fahringer, *Grid capacity planning with negotiation-based advance reservation for optimized QoS*, In Proceedings of the 2006 *ACM/IEEE conference on Supercomputing*, No. 103, November 2006.

Chapitre 5

Conclusion et perspectives

5.1. Conclusion

Dans ce mémoire, nous nous sommes intéressés aux problèmes de la réservation/allocation des ressources de la grille de calcul qui mènent à l'augmentation du blocage des demandes et par la suite aux pertes de revenus substantielles des fournisseurs des ressources/services. Nous avons proposé à cet égard deux contributions.

Dans la première contribution, nous avons donné la possibilité aux clients d'exprimer plus d'une qualité de service pour un type de ressources, et nous avons proposé un schéma d'adaptation qui maximise l'utilisation des ressources, réduit la probabilité de blocage et maximise le revenu des fournisseurs. L'idée de base derrière notre proposition consiste à ajuster les réservations, à la réception d'une nouvelle demande, au départ d'une demande existante ou à la dégradation des services, de façon à maximiser la quantité des ressources réservées et réduire le nombre de demandes rejetées en raison de l'indisponibilité des ressources.

Plus précisément, nous avons développé un modèle d'optimisation en utilisant la programmation en nombres entiers binaire (BIP). Ensuite, nous avons défini une heuristique (LOH) pour la résolution du modèle avec un temps de réponse et un nombre de mouvement acceptable. Une seconde heuristique GOH a été développée pour améliorer la solution courante en background. Les résultats des simulations ont montré que la combinaison de LOH et GOH (LOH+GOH) surpasse tous les autres techniques décrites dans le chapitre 3, en termes de revenus et de taux de rejet. LOH+GOH prévoit également une meilleure performance qu'une solution exacte mais avec un peu moins de revenus.

Notre approche reste applicable pour plusieurs ressources hétérogènes indépendantes. Il suffit de l'exécuter autant de fois qu'on a de ressources demandées.

Dans la deuxième contribution nous avons proposé une approche qui permet le provisionnement de ressources de calcul et de bande passante pour les applications et services de grille de calcul. L'idée derrière notre proposition c'est l'allocation du maximum possible des processus de même demande au même domaine; l'objectif est de minimiser l'utilisation de la bande passante et de réduire par la suite la probabilité de blocage dû au manque de bande passante et le taux d'arrêt brusque des sessions dû aux pannes réseaux.

Plus précisément, nous avons développé un modèle d'optimisation en utilisant la programmation en nombres entiers binaire (BIP). Ensuite, nous avons défini une heuristique (GIT) pour la résolution du modèle avec un temps de réponse acceptable, un minimum de coût d'utilisation des ressources, et un taux de rejet réduit. Les résultats des simulations ont montré que GIT produit un très bon temps de réponse en milliseconds et surpasse les techniques classiques en termes de coût et de taux de rejet. Notre méthode de recherche GIT peut être implémentée facilement avec d'autres structures de voisinages, telle que la permutation des processus entre deux domaines sans désignation dès le départ des processeurs qui vont entrer dans l'échange.

5.2. Perspectives

Parmi les développements futurs de ce projet qui pourraient être intéressants, nous prévoyons la considération de plusieurs types de ressources (plus que deux) dépendantes. Pour ce faire, nous pensons que le modèle de spécification des besoins devrait être conçu d'une façon à donner aux utilisateurs la possibilité d'exprimer les dépendances qui existent entre les différents composants de sa demande, en termes de temps, des attributs, de communication, de quantité de ressources, etc. Ensuite, la conception d'un modèle d'optimisation qui reflète la projection de ces dépendances en termes de dépendances entre les différents types des ressources considérées.

Bibliographie

[1] R. Al-Ali, A. Hafid, O. Rana, D. Walker, *An Approach for QoS Adaptation in Service-Oriented Grids, The Journal of Concurrency: Practice and Experience*, Vol. 16, Issue 5, Pages 401- 412 , 2004.

[2] R. Al-Ali, O. Rana, G. von Laszewski, A. Hafid, K. Amin, D. Walker, *A Model for Quality-of-Service Provision in Service Oriented Architectures, Journal of Grid and Utility Computing,* 2005.

[3] X. Bai, H. Yu, Y. Ji, D. C. Marinescu, *Resource Matching and a Matchmaking Service for an Intelligent Grid, ICCI,* Pages 257-261, December 2004.

[4] C. Barz, M. Pilz, T. Eickermann, L. Kirtchakova, O. Wäldrich, W. Ziegler, *Co-Allocation of Compute and Network Resources in the VIOLA-Testbed* 2006.

[5] I. Cardei, R. Jha,M. Cardei, A. Pavan., *Hierarchical architecture for real-time adaptive resource management. In IFIP/ACM International Conference on Distributed Systems Platforms*, Pages 415–434, 2000.

[6] B. Doshi, S. Wang, P. Kim; D. Goldsmith, B. Liebowitz, K. Park, *Cooperative Service Level Agreement, Military Communications Conference*, Pages 1-7, 23-25 Oct 2006.

[7] A. Filali, A. Hafid, M. Gendreau, *Adaptive Resources Provisioning for grid applications and services, IEEE ICC'08*, China, 2008.

[8] A. Filali, A. Hafid, M. Gendreau, *Bandwidth and Computing Resources Provisioning for Grid Applications and Services, IEEE GC'08*, 2008.

[9] I. Foster and C. Kesselman, eds., *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 2003.

[10] I. Foster, C. Kesselman, S. Tuecke, *The anatomy of the Grid enabling scalable virtual organizations, International J. Supercomputer Applications*, Vol. 15, No. 3, Pages 200-222, 2001.

[11] I. Foster, A. Roy, *A Quality of Service Architecture that Combines Resources Reservation and Application Adaptation. In Proceedings of 8th International Workshop on Quality of Service*, Pittsburgh, PA, Pages 181-188, 2000.

[12] P. Garbacki, V. K. Naik, *Efficient Resource Virtualization and Sharing Strategies for Heterogeneous Grid Environments; Integrated Network Management*, Pages 40 – 49, 2007.

[13] M. Gendreau, J.-Y Potvin, *Search methodologies: Introductory Tutorials in Optimization and Decision Support Techniques (edited by Edmund K. Burke, Graham Kendall)*, Ch. 6 "*Tabu-Search*", pages 165-186, *Springer*, 2005.

[14] F. Glover "*Future Paths for Integer Programming and Lines to Artificial Intelligence*", *Computer and Operations Research* 13, pages 533-549, 1986.

[15] F. Glover, *Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problem, Discrete Appl. Math.*, Vol 65, No.1- 3, Pages 223-253, 1996.

[16] G. Gutin, A. Yeo, A. Zverovich, *Introduction to Algorithms (Cormen, Leiserson, and Rivest)*, Ch 16 "*Greedy Algorithms*", 2001.

[17] A. Hafid, G. Bochmann, *Quality of Service Adaptation in Distributed Multimedia Applications, ACM Springer-Verlag Multimedia Systems Journal*, Vol. 6, No. 5, Pages 299- 315, 1998.

[18] ILOG CPLEX. http://www.ilog.com

[19] M. Karsten, N. Berier, L. Wolf, R. Steinmetz, *A Policy-Based Service Specification for Resource Reservation in Advance, In Proceedings of the International Conference on Computer Communications*, Tokyo, Japan, 1999.

[20] K. Kim, K. Nahrstedt, *A Resource Broker Model with Integration Reservation Scheme, in IEEE ICME*, NY, USA, Pages 859-862, 2000.

[21] B. Li, K. Nahrstedt, *A Control-Based Middleware Framework for Quality-of-Service Adaptations, IEEE,* Vol. 17 Issue 9, Pages 1632- 1650, 1999.

[22] J. Li, R.Yahyapour, *A Negotiation Model Supporting Co-Allocation for Grid Scheduling, The 7th IEEE/ACM International Conference on Grid Computing The*

7th IEEE/ACM International Conference on Grid Computing, Barcelona, Sept 28-29, *IEEE Computer Society Press*, 2006.

[23] S. Loudni, P. Boizumault, *Optimization sous contraintes en contexte anytime*, http://jedai.afia-france.org/repository/29.pdf.

[24] V. K. Naik, P. Garbacki, Krishna Kummamuru, Y. Zhao, *On-line Evolutionary Resource Matching for Job Scheduling in Heterogeneous Grid Environments, In Proceedings of the 12th International Conference on Parallel and Distributed Systems*, Pages 103-108, 2006.

[25] V. K. Naik, C. Liu, L. Yang, J. Wagner, *Online resource matching for heterogeneous grid environments, In Proceedings of 5th IEEE International Symposium on Cluster Computing and the Grid*, Vol. 2 Pages 607-614, 2005.

[26] N. Nehra, R. B. Patel, V. K. Bhat, *Distributed Parallel Resource Co-allocation with Load Balancing in Grid Computing, IJCSNS*, Vol. 7, No.1, Pages 282-292, 2007.

[27] D. A. Reed, C. L. Mendes, *Intelligent Monitoring for Adaptation in Grid Applications, In Proceedings of the IEEE*, Vol. 93, Issue 2, Pages 426- 435, 2005.

[28] R. L. Ribler, J, S. Vetter, H Simitci, D. A. Reed, *Autopilot: Adaptive Control of Distributed Applications, 7th IEEE Symp. On High Performance Distributed Computing*, 1998.

[29] A. Roy, V. Sander, *GARA: a uniform quality of service architecture, Grid resource management: state of the art and future trends, Kluwer Academic Publishers*, Norwell, MA, Pages 377-394, 2004.

[30] M. Siddiqui, A. Villaz, T. Fahringer, *Grid capacity planning with negotiation-based advance reservation for optimized QoS, In Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, No. 103, November 2006.

[31] M. Yagiura, T. Ibaraki, F. Glover, *An Ejection Chain Approach for the Generalized Assignment Problem*, Vol 16, Pages 133-151, 2004.