

A study of meta-scheduling architectures for high throughput computing

Eddy Caron, Vincent Garonne, Andreï Tsaregorodtsev

► **To cite this version:**

Eddy Caron, Vincent Garonne, Andreï Tsaregorodtsev. A study of meta-scheduling architectures for high throughput computing. [Research Report] LIP RR-2005-13, Laboratoire de l'informatique du parallélisme. 2005, 2+13p. hal-02101915

HAL Id: hal-02101915

<https://hal-lara.archives-ouvertes.fr/hal-02101915>

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

***A study of meta-scheduling architectures
for high throughput computing***

Eddy Caron ,
Vincent Garonne ,
Andrei Tsaregorodtsev

March 2005

Research Report N° 2005-13

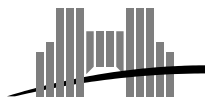
École Normale Supérieure de Lyon

46 Allée d'Italie, 69364 Lyon Cedex 07, France

Téléphone : +33(0)4.72.72.80.37

Télécopieur : +33(0)4.72.72.80.80

Adresse électronique : lip@ens-lyon.fr



A study of meta-scheduling architectures for high throughput computing

Eddy Caron , Vincent Garonne , Andreï Tsaregorodtsev

March 2005

Abstract

In this paper we present a model and a simulator for large-scale system. Such platforms are composed of heterogeneous clusters of PCs belonging to a local network. These clusters are then connected to each other through a global network. Moreover each cluster is managed via a local scheduler and is shared by many users. We validate our simulator by comparing the experimental results and the analytical results of a M/M/4 queuing system. These studies indicate that the simulator is consistent. After that we do the comparison with a real batch system and we obtain a mean error of 10.5 % for the response time and 12 % for the makespan. We conclude that our simulator is realistic and describes well the behavior of a large-scale system. Thus we can study the scheduling of our system called *DIRAC* in a high throughput context. We justify our decentralized, adaptive and opportunistic approach in comparison to a centralized approach in such a context.

Keywords: Metascheduler, Grid computing

Résumé

Dans cet article, nous présentons une modélisation et un simulateur de grands systèmes de calcul distribué. Une telle plate-forme se compose de grappes de PCs hétérogènes appartenant à un réseau local inter-connectées entre elles par un réseau global. Ces grappes sont accessibles via un ordonnanceur local et sont partagées entre les utilisateurs. La confrontation du simulateur avec les résultats théoriques d'un système M/M/4 nous permet de conclure qu'il est analytiquement valide. Une deuxième confrontation avec un système batch réel, nous donne une différence moyenne de 10.5 % par rapport à la réalité pour les temps de réponse et de 12% pour le makespan. Notre simulateur est donc réaliste et décrit le comportement d'un système de batch réel. Fort de cet outil, nous avons analysé l'ordonnancement de notre système (appelé *DIRAC*) dans un contexte de calcul intensif. Nous avons justifié l'approche distribuée, adaptative et opportuniste utilisée dans notre système par rapport à une approche centralisée.

Mots-clés: Plate-forme Multi-grappes, Méta-Ordonnancement, Grilles de calcul

1 Introduction

Dans les grands systèmes de calcul distribués [6] et institutionnel, les ressources de calcul sont des grappes hétérogènes de PCs appartenant à un réseau local et divers domaines administratifs. Ces grappes sont partagées entre différents utilisateurs ou organisations virtuelles [4]. Ainsi une politique locale à chaque grappe définit le partage et l'accès entre les différentes organisations. Cette politique s'applique directement à travers le système local de gestion des ressources, i.e. un système de Batch.

Pour fédérer ces grappes et gérer la charge de calcul globale, la définition d'une architecture globale est vitale. De plus la taille de ces systèmes est importante. Par exemple, dans le domaine des nouvelles expériences de la physique des particules, l'ordre de grandeur envisagé se situe autour d'une centaine de grappes réparties dans le monde et représentant 30.000 nœuds. L'autre caractéristique de ce domaine est le contexte de calcul intensif (*High Throughput Computing*) [12]. Nous nous intéressons alors à l'utilisation maximale du système sur une longue période. Le calcul intensif favorise le régime permanent et saturé du système. Bien qu'au niveau local l'utilisation de système de batch est fréquente, il n'existe pas d'architecture standard au niveau global pour le calcul intensif. Le système *DIRAC* propose justement dans ce cas précis un exemple de solution à ce manque. De ce fait, nous proposons dans cet article une étude pour analyser les performances et le comportement de *DIRAC* dans une situation de calcul intensif.

Après avoir abordé les travaux liés au paradigme *push* dans la section 2, la section 3 présente le système *DIRAC* et plus particulièrement la gestion de la charge dans ce système. Nous décrivons ensuite, dans la section 4, la modélisation d'une plate-forme de méta-computing que nous avons réalisé. La complexité de ces systèmes est telle que beaucoup de difficultés demeurent pour évaluer l'impact et les bénéfices d'une stratégie d'ordonnancement. Nous avons donc ici recours à des simulations. Le simulateur que nous avons développé dans ce but est présenté dans la section 5. On y parlera également des apports et ajouts aux primitives définies par *Simgrid* [1]. La validation de ce simulateur avec l'infrastructure *DIRAC* est exposé en section 6. En section 7, il nous est alors possible de comparer l'approche décentralisée *DIRAC* avec une approche centralisée. Les résultats obtenus par le simulateur sont présentés en section 8.

2 Travaux reliés: le paradigme 'Push'

Pour compléter cette introduction il est intéressant de mentionner les travaux liés au paradigme 'Push'. Dans les projets de grille actuels [18, 8] sur l'ordonnancement multi-sites, les décisions se basent sur une connaissance totale de l'état du système à l'instant t . Les architectures sous-jacentes consistent en un méta-ordonnanceur centralisé qui se base sur un système d'informations centralisé. Cette approche, illustrée figure 1, est qualifiée de méthode 'Push'.

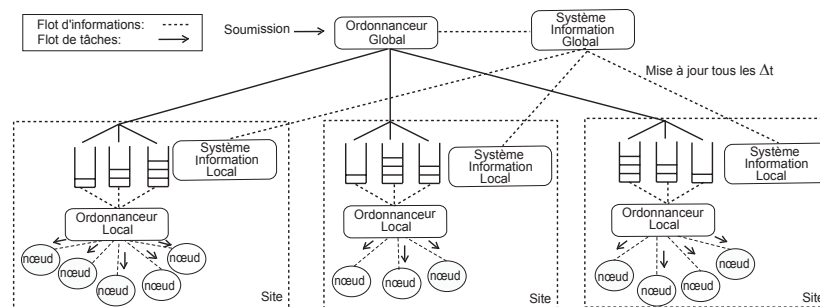


Figure 1: Exemple d'architecture avec un ordonnanceur centralisé.

Le système d'information global contient les informations dynamiques et statiques sur l'état de la plate-forme. Les informations dynamiques sont mises à jour par des senseurs déployés sur les sites. Ils interrogent le système d'information local puis positionnent les informations relatives au

site dans le système d'information global. Une des difficultés d'un système réel est de maintenir une vue de l'image de l'ensemble de la plate-forme la plus réaliste possible. Idéalement un agent effectue une mise à jour dès que l'état du site subit un changement. Ce changement correspond à la fin d'une tâche ou à l'arrivée d'une tâche. Mais concrètement cette solution entraîne un surcoût important de messages et nécessite des mécanismes de notification. L'utilisation d'une période Δt de rafraîchissement de l'information est un compromis entre cohérence des informations et nombre de messages.

Des études [9] proposent des stratégies qui utilisent des systèmes à files d'attente [10]. D'autres [17] ont souvent recours à des simulations comme BRIKS [16]. Généralement, ces études permettent d'estimer des stratégies mais ceci reste dans des modèles simplifiés et souvent non réalistes.

À notre connaissance, aucun projet n'a encore réussi à répartir efficacement la charge sur une centaine de sites et la problématique d'un système multi-sites pour le calcul intensif dans un régime permanent et saturé n'a pas été traité.

3 Le système *DIRAC*

Le système *DIRAC* [7] est un environnement distribué. Il est utilisé intensivement comme système d'analyse et de production de données, via des simulations Monte-Carlo pour les expériences de physique des hautes énergies *LHCb* (CERN). Pendant les 4 derniers mois *DIRAC* a utilisé une soixantaine de centres de calcul et plus de 4.000 nœuds pour exécuter 200.000 tâches [3]. 70 To de données ont été produites, répliquées et stockées. Le système repose sur une *architecture orientée service*, illustrant ainsi la convergence des systèmes de calcul global, pair-à-pair et des systèmes de grid computing [5]. Cette approche vise à avoir une grande souplesse d'intégration, de remplacement ou d'évaluation de services. La philosophie de *DIRAC* est d'être un système léger, extensible et robuste. Il fournit ainsi un ensemble de services pour un système de méta-computing générique. Chaque service est composé de fonctionnalités et d'interfaces précises. Parmi les services, citons le service de monitoring et le service de gestion de charge. Ce dernier est un service vital, nous allons en donner une description détaillée.

3.1 Le service de gestion de charge: le paradigme 'Pull'

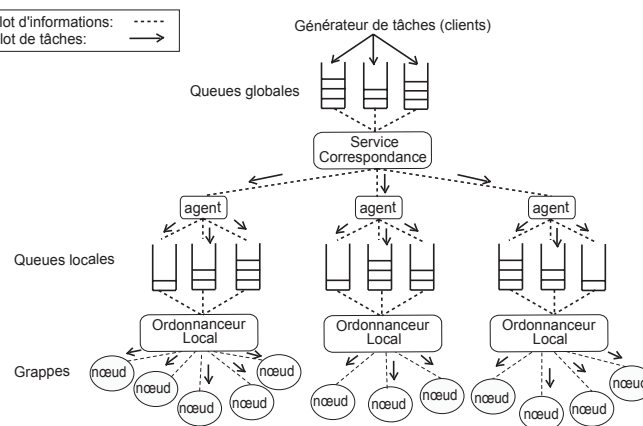


Figure 2: Le Modèle d'ordonnancement *DIRAC*.

La figure 2 illustre l'architecture de l'ordonnancement de *DIRAC*. Il met en place des files d'attente centrales et des agents déployés sur des centres de calcul. *DIRAC* utilise un paradigme 'Pull' où le rôle des agents est de demander des tâches quand ils détectent la disponibilité des ressources locales. *DIRAC* emprunte cette idée des systèmes de calcul global ou de vols de cycles [14], dans lesquels un serveur central distribue des tâches à des nœuds en fonction de leurs

disponibilités. *DIRAC* transpose ce concept au niveau de ressources de calcul distribuées en définissant un critère de disponibilité. Ces ressources peuvent être de simples ordinateurs, des systèmes de batch.

Dès qu'une ressource est détectée comme disponible, l'agent dédié à cette ressource demande une tâche à un service correspondance. Cette requête s'effectue avec la description de la ressource, qui indique l'état dynamique et statique de la ressource. Le service correspondance résout la correspondance entre la description de la ressource et les tâches disponibles en file d'attente. Le langage utilisé pour décrire les ressources, les tâches et les opérations de correspondance est *Classad* du projet *CONDOR* [12].

Pour diminuer la complexité des opérations de correspondance, les tâches sont regroupées par caractéristiques et besoins. Ces catégories définissent des classes de tâches. Cette opération a donc une complexité de $O(n)$ ou n est le nombre de classes de tâches. Cette opération est indépendante du nombre de ressources dans le système et aussi du nombre de tâches.

4 Modélisation

Considérons une plate-forme de méta-ordonnancement constituée de \mathcal{C} grappes de processeurs. Chaque grappe \mathcal{C}_i possède un ensemble de nœuds de calcul \mathcal{N}_i .

4.1 La topologie

Chaque grappe \mathcal{C}_i appartient à un domaine local, i.d. un LAN(Local Area Network). Ce réseau local décrit un graphe d'interconnexion des nœuds. Chaque lien de ce graphe est pourvu d'**une bande passante locale** $bwt_{\mathcal{C}_i}$ d'**une latence locale** $latence_{\mathcal{C}_i}$. La connexion d'une grappe \mathcal{C}_i au réseau global ou WAN(World Area Network) se fait par un routeur. Un graphe décrit aussi cette interconnexion avec des liens ayant les mêmes propriétés que précédemment soit **la bande passante globale** $bwt_{\mathcal{C}}$ et **la latence globale** $latence_{\mathcal{C}}$.

Nous observons ainsi une topologie hiérarchique comme l'illustre la figure (3).

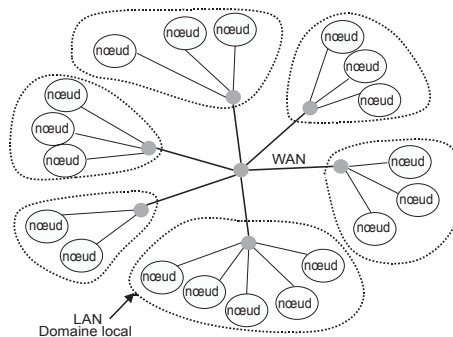


Figure 3: Exemple d'une topologie pour une plate-forme multi-grappes.

Pour générer le graphe conforme à la modélisation énoncée, différentes approches existent. Une méthode consiste à décrire une topologie réelle. Pour ceci des outils de découvertes de caractéristiques comme ENV [15] existent. Le passage à l'échelle de ces outils n'étant pas garanti nous avons décidé de privilégier des générateurs de topologie. Des études récentes [13] montrent que les réseaux actuels suivent des lois particulières. Les générateurs de graphes selon ces lois sont principalement de trois types: aléatoire, basé sur le degré ou hiérarchique.

Soit le couple (i, j) définissant le $j^{i\text{me}}$ nœud d'une grappe \mathcal{C}_i . Chaque nœud (i, j) est caractérisé par une puissance $puissance_{i,j}$ du processeur. Pour exprimer cette puissance, définissons une unité le **NCU** (Normalized Computing Unit) qui équivaut à une unité de calcul. Cette unité se détermine notamment par des benchmarks d'une application précise sur différents types de machines en fonction du temps d'exécution absolu sur une machine référence. Ainsi la puissance

d'un nœud est le nombre d'unités de calcul traitées par unité de temps. Nous pouvons ensuite exprimer l'hétérogénéité de la plate-forme et désigner **la puissance moyenne** de la plate-forme par $puissance_m = \frac{1}{\sum_{i \in \mathcal{C}} card(\mathcal{N}_i)} \sum_{i \in \mathcal{C}, j \in \mathcal{C}_i} puissance_{i,j}$.

4.2 Le modèle de charge

Nous distinguons deux niveaux de charge, une locale et l'autre globale. La charge globale correspond aux tâches soumises par le système de méta-computing. Ces tâches sont appelées méta-tâches. La charge locale composée de tâches est la charge propre à une grappe. Une méta-tâche m_k est traduite au niveau local en une simple tâche k .

Une tâche k est caractérisée par quatre attributs : $attributs_k = \{tl_k, taille_k, proc_k, group_k\}$ où tl_k est **la date de soumission locale** sur un site, $taille_k$ **la taille** exprimée en NCU, $proc_k$ **le nombre de processeurs** requis pour son exécution et $group_k$ **l'organisation** qui soumet la tâche. Une méta-tâche mk est aussi caractérisée par $meta-attributs_k = \{t_k, attributs_k\}$ où t_k est **la date de soumission** de la tâche mk au système de méta-computing global.

Modéliser la charge d'un système de méta-computing revient à déterminer pour chaque tâche k de l'ensemble des tâches \mathcal{T} soumises à une grappe \mathcal{C}_i les caractéristiques $attributs_k$. Il en va de même pour chaque méta-tâche mk de l'ensemble de tâches \mathcal{MT} soumises au système de méta-computing et leurs méta-attributs. Les méthodes pour générer une charge sont de trois types : une charge aléatoire, une charge issue des traces d'un système réel ou une charge stochastique. Une charge aléatoire est peu réaliste de même que celles issues de systèmes réels sont souvent relatives à la capacité et le temps de réponse du système réel et donc trop spécifique. Une charge stochastique est souvent la plus réaliste et sera notre choix. Des travaux étudiant les traces de centres de calcul [11] proposent des modèles probabilistes complets. Nous notons $S(\mathcal{T})$ **la fonction de distribution qui génère l'ensemble des tailles** d'un ensemble de tâches \mathcal{T} . Soit CA **la coupure appliquée** à cet ensemble de tailles fixant la taille minimale et maximale. Nous désignons par $A(\mathcal{T})$ **la fonction de distribution qui génère l'ensemble des temps de soumission** d'un ensemble de tâches et $\lambda_{\mathcal{T}}$ **le taux de soumission moyen**.

4.3 Les architectures et modèles d'ordonnancement local

Au niveau local, les nœuds d'un même site sont typiquement gérés par un système de gestion des ressources comme un système de batch. Sur le marché la plupart des solutions existantes mettent toutes place des ordonnanceurs utilisant des files d'attente (ou queues). Ces files d'attente sont souvent définies en fonction des caractéristiques des tâches par exemple leurs tailles. Ces ordonnanceurs locaux régissent le partage des ressources entre utilisateurs par des politiques locales reposant sur des stratégies de quotas et de priorités. Ainsi nous avons pour une grappe \mathcal{C}_i , **un ensemble de files d'attente** \mathcal{F}_i . Chaque file $\mathbf{f}_{i,j}$ de \mathcal{F}_i est composée d'un ensemble de nœuds $\mathbf{N}_{\mathbf{f}_{i,j}}$ de la grappe. Un même nœud peut appartenir à une ou plusieurs files. Les tâches soumises au site sont ensuite affectées à ces files en attendant leur exécution en fonction de la stratégie d'ordonnancement locale basée sur leurs priorités et contraintes. De ce fait, une file $\mathbf{f}_{i,j}$ contient un ensemble de tâche $\mathcal{T}_{i,l}$ en attente d'exécution. Nous définissons aussi **la profondeur de la file d'attente** comme $profondeur_{i,j} = card(\mathcal{T}_{i,l})$ qui est **le nombre de tâches en attente** dans la file à un instant. Nous notons $t_{max_{i,j}}$ **le temps maximal** d'une tâche en exécution sur un nœud de la file d'attente $\mathbf{f}_{i,j}$.

4.4 États, mesures et métriques

Nous définissons pour une tâche k , les trois états suivants: *queued*, *running*, *done*. L'état *queued* signifie que la tâche est en file d'attente. Quand la tâche est en exécution elle est en état *running*. L'état *done* indique que la tâche a fini son exécution. **Les dates correspondantes aux changements d'états *running*, *queued* et *done* pour une tâche k sont respectivement r_k, q_k et d_k** . Ainsi **le temps d'attente local d'une tâche k** est la date de début d'exécution

moins la date de soumission soit $r_k - tl_k$, le **temps d'exécution** est $d_k - r_k$ et le **temps de réponse local** est $d_k - tl_k$.

Pour une méta-tâche mk , nous avons le **temps d'attente global** qui correspond à la date de début d'exécution moins la date de soumission au système soit $r_k - t_k$ et le **temps de réponse global** est $d_k - t_k$.

Pour l'ensemble de méta-tâches \mathcal{MT} , nous définissons le **temps d'attente moyen**:

$$attente_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (r_k - t_k) \quad (1)$$

le **temps d'exécution moyen**:

$$execution_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (d_k - r_k) \quad (2)$$

et le **temps de réponse moyen**:

$$moyen_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (d_k - t_k) \quad (3)$$

Nous définissons aussi le **makespan** qui est le temps total pour terminer l'exécution de toutes les tâches dans l'ensemble \mathcal{MT} :

$$makespan = \max_{k \in \mathcal{MT}} (d_k) - \min_{k \in \mathcal{MT}} (t_k) \quad (4)$$

5 L'outil de simulation

Simgrid [1] est un outil à évènement discret fournissant des primitives qui permettent de modéliser et de décrire une plate-forme pour l'ordonnancement centralisé, hiérarchique ou décentralisé.

Par rapport à la modélisation décrite en section 4, les entités implémentées dans le simulateur sont les suivantes:

un module de description de la plate-forme. Notre simulateur est interfacé avec le générateur de graphe *Tiers* [2]. Il faut spécifier le nombre de WAN, le nombre de LAN, le nombre de nœuds par LAN et le nombre de liens redondants. Pour l'attribution de la puissance, nous définissons un échantillon de nœuds de référence. Chaque nœud est pondéré par un pourcentage. Ce pourcentage exprime la proportion de ce type de nœud présent sur la plate-forme. La puissance NCU des nœuds et leurs pondérations s'inspirent des performances obtenues avec *DIRAC* pour une application de physique sur la plate-forme de production [3]. Cette plate-forme se composait de 4.000 nœuds et d'une vingtaine de configurations de nœuds différents. En fonction du nombre total de nœuds et des proportions attribuées à chaque nœud de référence, nous générons l'ensemble P de toutes les puissances disponibles. Ensuite pour chaque nœud nous procédons à un tirage aléatoire dans cet ensemble. Nous otions ensuite une occurrence de cette valeur de l'ensemble P et ce jusqu'à l'attribution de la puissance pour tous les nœuds. Nous disposons ainsi d'une plate-forme hétérogène.

un générateur de charges. *Simgrid* implémente déjà le concept de tâche. Nous avons rajouté les méta-données supplémentaires de notre modèle comme l'organisation soumettant la tâche. Nous avons aussi mis en œuvre un générateur de charges. Celui-ci génère des charges en fonction de différentes lois de distribution comme des lois de gamma, normale ou autres. De plus, pour avoir un système partagé nous avons implémenté un agent qui représente soit un client du système ou une organisation et qui soumet une charge spécifique. Ce simulateur permet donc de simuler le comportement des utilisateurs du système. Nous pouvons avoir ainsi un système soumis à plusieurs charges différentes en même temps et évaluer les interactions.

un système de batch générique. Au niveau local, l'entité de base d'un système de méta-ordonnancement est le système de batch. *Simgrid* ne le fournissant pas, nous avons implémenté un système de batch générique. Le modèle de conception utilisé est illustré sur la figure 4.

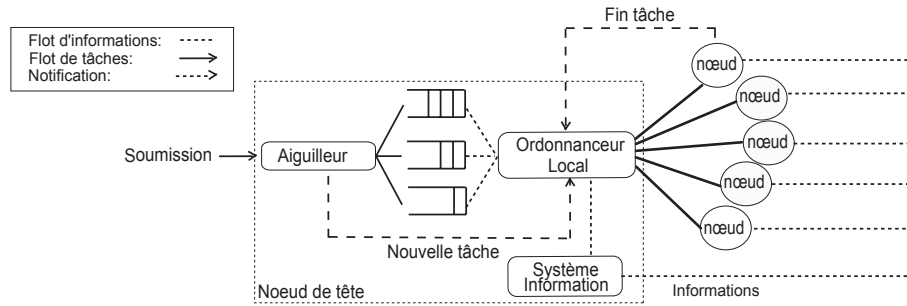


Figure 4: Structure d'un système de Batch générique.

Un nœud de tête héberge les composants principaux de l'ordonnanceur local: l'aiguilleur, les files d'attente, le système d'information puis l'ordonnanceur local. Chaque nœud contient un composant qui communique avec le nœud de tête. La soumission d'une tâche est traitée par un aiguilleur qui en fonction des besoins exprimés dans les méta-données de la tâche affecte celle-ci à une file d'attente. Il notifie ensuite l'ordonnanceur. Celui-ci interrogera le service de monitoring et de comptage local pour ensuite élire un nœud candidat. Si aucune ressource n'est disponible, la tâche reste en file d'attente. Après l'envoi de la tâche au nœud, la tâche est exécutée. À la fin de l'exécution de la tâche l'ordonnanceur est notifié. Ceci enclenche un cycle d'ordonnancement, l'ordonnanceur examine les files d'attente et détermine si une tâche est apte à être exécutée. La configuration locale de chaque ordonnanceur est paramétrable par fichier. Cette configuration comprend le nombre de files d'attente, leurs caractéristiques, l'appartenance d'un nœud à une ou plusieurs files ou le nombre de tâches pouvant s'exécuter sur un nœud.

Les architectures globales Nous avons implémenté deux types d'architectures globales. Une architecture centralisée détaillée dans la section 1 et celle de *DIRAC* décrite dans la section 3.

un système de monitoring et de comptage des tâches local et global. Un système de monitoring et de comptage local contient une base de données qui contient l'état courant des nœuds. Cette base est mise à jour par les nœuds du système, l'aiguilleur et l'ordonnanceur. Elle contient aussi tous les changements d'état des tâches soumises au site et les informations relatives à leurs exécutions. Cette base comporte aussi le comptage des tâches exécutées sur le système et les méta-données correspondantes. Pour pouvoir analyser les résultats, nous enregistrons pour chaque expérimentation les informations de chaque tâche. Ces informations aident l'analyse d'une stratégie en permettant le calcul des mesures et métriques définies en section 4.4.

6 Validation du simulateur

Après la conception du simulateur, pour déterminer la pertinence de notre outil une phase de validation est primordiale. Le simulateur a été validé de manière théorique puis expérimentale.

6.1 Validation théorique

Pour la validation théorique de notre simulateur, nous utilisons ici un système à file d'attente de type M/M/m où le premier M désigne la distribution des intervalles de temps séparant deux

arrivées successives de tâches (les inter-temps), le second M est la distribution du temps de service, et m dénote le nombre de processeurs [10].

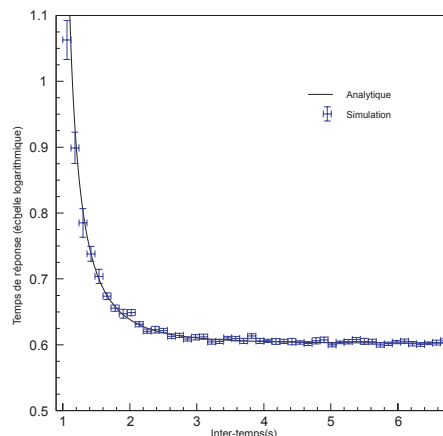


Figure 5: Comparaison des temps de réponses entre la simulation et la théorie dans un système de file d'attente M/M/4.

La figure 5 montre la confrontation des temps de réponses analytiques et simulés dans un système M/M/4 en fonction des inter-temps. Les temps d'exécution sont distribués exponentiellement avec une moyenne de 4 unités de temps. Les inter-temps sont aussi distribués exponentiellement. Les temps de réponses simulés sont obtenus par la méthode des moindres carrés avec 16 exécutions indépendantes de 1000 tâches pour une moyenne d'inter-temps donnée. Les résultats entre la simulation et la théorie coïncident. Nous concluons ainsi que notre simulateur est validé du point de vue théorique.

6.2 Validation expérimentale du simulateur

Nous avons utilisé une grappe de PCs dédiée et hétérogène dont la configuration est résumée dans le tableau 6.2.

Attributs	Valeurs		
Nombre de nœuds($card(\mathcal{N}_i)$)	3		
Type	PII	PII	PIII
Processeur(MHz)	350	400	600
Mémoire vive(mo)	128	128	128
Puissance($NCU.s^{-1}$)	32.12	52.12	100.00
Ordonnanceur	openPBSv2.3		
Politique	First Come First Serve(FCFS)		
Réseau local	Mégabit Ethernet		

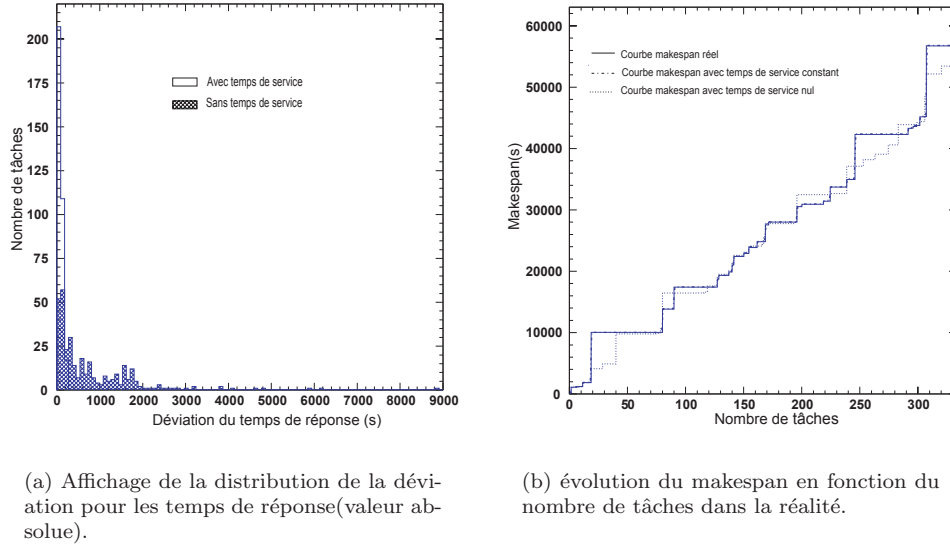
Table 1: Résumé des caractéristiques de la plate-forme utilisée pour la validation du simulateur.

Nous avons utilisé le système *DIRAC* pour valider notre simulateur. Nous avons déployé un agent *DIRAC* sur le site et utilisé un générateur de tâches. Ce générateur génère et soumet des tâches de calcul séquentielles indépendantes et donc sans communication. Les temps de soumission suivent une loi de distribution de Poisson.

Le jeu d'essai utilisé est un programme qui implémente un compteur de consommation CPU. Il prend un paramètre qui est le temps CPU à consommer avant de se terminer. Cette taille est générée pour chaque tâche et suit une loi de Weibull. Les temps de réponse et d'attente du système

sont recueillis par le service de monitoring de *DIRAC*. Nous dérivons ensuite cette charge pour l’injecter dans notre simulateur. Pour décorrélérer le temps d’exécution du nœud sur lequel une tâche s’exécute nous normalisons ce temps avec la puissance NCU du nœud. La puissance NCU des nœuds est obtenue par des jeux d’essai et est reportée dans le tableau 6.2.

La topologie utilisée est une topologie simple ou chaque nœud est relié au nœud de tête comme illustré sur la figure 4 par un simple lien d’une bande passante de 100 Mbit par secondes et de latence nulle.



(a) Affichage de la distribution de la déviation pour les temps de réponse (valeur absolue).

(b) évolution du makespan en fonction du nombre de tâches dans la réalité.

Figure 6: Confrontation entre la simulation et la réalité, avec $\mu_{rec} = \mu_{env} = 0$ et $\mu_{rec} = constante$, $\mu_{env} = constante$.

La figure 6(a) montre deux distributions de la valeur absolue de la déviation observée entre la réalité et la simulation pour les temps de réponse de chaque tâche. Le nombre de tâches est de 330, i.d. $card(\mathcal{MT}) = 330$. Dans la première distribution nous observons un fort taux d’erreur de l’ordre de 80%. Après une étude des traces, nous avons caractérisé deux temps de service μ_{rec} et μ_{env} . μ_{rec} est le temps de service entre l’arrivée d’une tâche et l’envoi de cette tâche sur un nœud ou en file d’attente. μ_{env} est le temps de prise en compte de la fin d’une tâche par l’ordonnanceur et de disponibilité d’un nœud. Ce fort taux d’erreur se comprend par le fait que l’ordonnanceur effectue ces choix avec des données différentes que dans la réalité. Les nœuds étant hétérogènes ceci a des conséquences immédiates sur le temps de réponse d’une tâche. Nous corrigeons ce taux d’erreur en incluant les temps de services mesurés sur le système réel pour les paramètres μ_{rec} et μ_{env} sous forme de traces. Nous obtenons avec ceci exactement le comportement de la réalité. Ceci valide le code de notre simulateur. Nous avons réitéré l’expérience en positionnant les temps de service à des constantes. Ces constantes sont les temps de services moyens mesurés sur le système test ($\mu_{rec} = 3.75s$, $\mu_{env} = 2s$). Ces paramètres diminuent le taux d’erreur car nous observons un taux d’erreur moyen de 10.5% pour le temps de réponse.

La figure 6(b) montre l’évolution du makespan en fonction du nombre de tâches en abscisse. Pour les temps de services constants, nous observons un taux d’erreur moyen de 12%. Notre simulateur au niveau local est donc réaliste. Nous pouvons ainsi procéder à l’évaluation de stratégies et d’architectures de méta-computing. Notons, qu’une possible amélioration serait de faire suivre pour les paramètres de temps de services μ_{rec} et μ_{env} une loi de distribution qui approche le comportement réel.

7 Expérimentations

Nous comparons l'architecture décentralisée de *DIRAC* décrite en section 3 et l'approche centralisée de la section 1 sur une plate-forme dédiée puis partagée. La taille des messages de contrôle dans la simulation est de 30 ko pour les deux architectures. Les caractéristiques des charges sont inspirées de l'étude empirique [11]. Le tableau 2 résume les paramètres de la plate-forme et des charges.

	<i>Paramètres</i>	<i>Notations</i>	<i>Valeurs</i>	
Plate-forme	Nombre de sites	$card(C)$	3	
	Nombre de nœuds par site	$card(N_i)$	20	
	Nombre de files d'attente par site	$card(F_i)$	1	
	Puissance moyenne de nœuds	$puissance_m$	$96 NCU.s^{-1}$	
	Politiques Locales	M/M/ $card(N_i)$ /FCFS	PAPS	
	Temps maximal d'exécution	$t_{max_{i,j}}$	24000s	
	Bande passante locale/globale	bwt_C/bwt_{C_i}	1000 Mbit/100 Mbit	
	Latence locale/globale	$latence_C, latence_{C_i}$	0s	
Charge	Globale	Type de tâches	$card(proc_k)$	1
		Distribution des tailles	$S(Mt) \rightarrow \{taille_k\}$	$Weibull(\alpha = 142.2, \beta = 0.45)$
		Coupure sur les tailles	$\mathcal{C}(taille_k)$	$37300 < taille_k < 242800$
		Nombre de tâches	$card(Mt)$	500
	Locale	Distribution des temps d'arrivées	$A(Mt) \rightarrow \{tl_k\}$	$Poisson(m = 0.05, s = 4)$
		Inter-temps moyen	$1/\lambda_{Mt}$	19s
		Nombre de tâches par site	$card(T_i)$	500
		Distribution des temps d'arrivées	$A(T_i) \rightarrow \{t_k\}$	$Poisson(m = 0.011, s = 4)$
	Inter-temps moyen	$1/\lambda_{T_i}$	87s	

Table 2: Résumé des paramètres pour les expériences.

La stratégie associée à l'architecture *DIRAC* est aussi celle décrite en section 3. Le critère de disponibilité choisi est donné en (5) et s'applique au niveau des files d'attente $f_{i,j}$ d'un cluster C_i .

$$\frac{profondeur_{i,j}}{card(\mathcal{N}_{f_{i,j}})} < \varepsilon, \text{ e.g. } \varepsilon = 0.3 \quad (5)$$

La politique appliquée au niveau du service **correspondance** est qualifiée de FRFS (Fit Ressource First Serve). C'est-à-dire que la première ressource qui convient à une tâche est choisie.

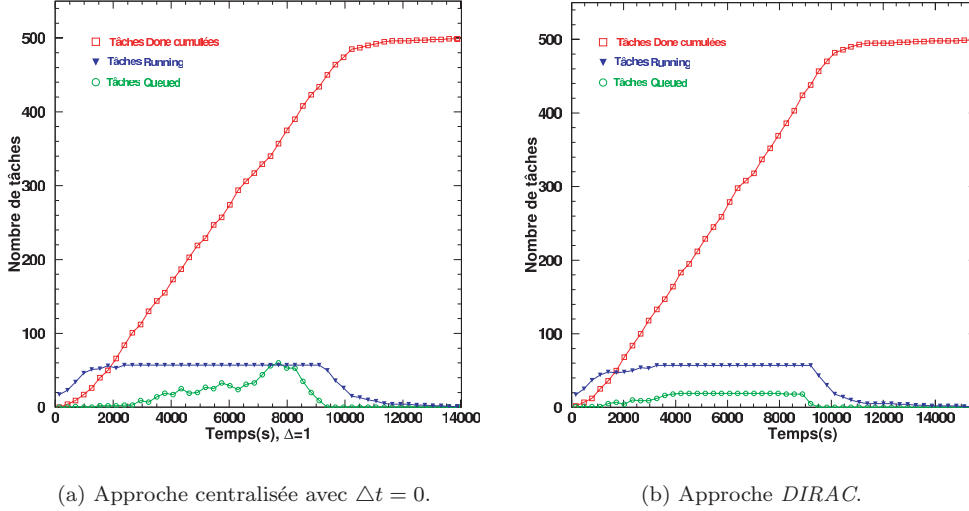
Nous avons voulu aussi évaluer l'importance du déploiement au sein de *DIRAC*. Pour ceci, nous considérons deux déploiements pour les agents. Un déploiement statique où l'agent déployé sur le site soumet des tâches dans les files d'attente. Ce cas est identique à celui expliqué en section 3 et l'autre déploiement est qualifié de dynamique ou de réservation. L'agent déployé sur le site détecte si la ressource est disponible. Dans le cas positif, il interroge le service correspondance pour savoir si des tâches sont disponibles. Si tel est le cas, il soumet des agents à la ressource et réitère sa demande. Ensuite, l'agent sur le nœud demande du travail au service correspondance utilisant la description du nœud générée au préalable. Si une tâche est disponible, l'exécution de cette tâche est effectuée. Dans le mode de réservation simple l'agent meurt après l'exécution d'une tâche. En mode de réservation *filling*, il récupère encore une tâche en accord avec les contraintes de temps restant. En cas de non disponibilité d'une tâche, l'agent meurt immédiatement après avoir reçu une réponse négative du service correspondance.

L'algorithme utilisé pour l'ordonnancement centralisé est le suivant: à chaque réception de tâche l'ordonnanceur affecte la ressource la moins chargée, c'est-à-dire la ressource $f_{i,j}$ du cluster C_i possédant la mesure $profondeur_{i,j}$ minimum avec $\forall i \in \mathcal{C}$ et $\forall j \in \mathcal{F}_i$.

Nous décrivons ici deux approches, une centralisée et l'autre décentralisée qui favorisent l'utilisation du système. Il nous reste à déterminer quelle influence peuvent avoir leurs implémentations et leurs architectures sur les performances.

8 Résultats et discussions

La figure 7 montre l'évolution du cardinal des tâches dans les différents états *queued* et *running* pendant la durée de l'expérience pour une plate-forme dédiée. La troisième courbe est la courbe cumulée de toutes les tâches en états *done*.



(a) Approche centralisée avec $\Delta t = 0$.

(b) Approche *DIRAC*.

Figure 7: Évolution du cardinal des états des tâches dans le temps dans le cadre d'une plate-forme dédiée.

La figure 7(b) montre ceci pour l'ordonnancement centralisé avec $\Delta t = 0$ et la figure 7(a) pour *DIRAC*. Pour les deux approches nous observons une phase d'initialisation du système puis la saturation de toutes les ressources et finalement l'arrêt du système. La phase de saturation nous donne la capacité maximale de la plate-forme. Il correspond à la somme de tous les nœuds disponibles soit $\sum_{i \in \mathcal{C}} \mathcal{N}_i$, ici 60. Les deux approches saturent donc bien les ressources. La caractéristique de l'approche *DIRAC* par rapport à l'approche centralisée est que le nombre de tâches en état *queued* est constant.

La figure 8(a) montre l'impact de la variation de la période Δt pour l'ordonnancement centralisé sur le temps d'attente moyen $attente_m$ dans un contexte dédié puis partagé. Nous avons effectué pour ceci une variation du paramètre Δt avec un pas de 5 s. Les temps d'attente moyens obtenus pour *DIRAC* sont donnés à titre indicatif car ceux-ci ne dépendent pas de Δt (figure 8(b)). *DIRAC* n'utilise pas de système d'information centralisé et ne dépend donc pas de la fréquence de rafraîchissement de l'information. Pour un Δt inférieur à 95 s, le temps d'attente moyen est meilleur pour l'ordonnancement centralisé dans un contexte dédié et à peu près inférieur à 60 s dans un contexte partagé, par contre il se dégrade rapidement. L'effet est plus chaotique dans un contexte partagé. Le plateau observé donne la borne supérieure qui correspond à la situation où toutes les tâches sont affectées au même site soit dans la situation où $\Delta t > \max_{k \in \mathcal{M}t} t_k$.

La figure 9(a) compare le makespan, le temps de réponse moyen local et global ainsi que le temps d'exécution pour les quatre stratégies évaluées.

Pour un Δt nul, le meilleur makespan est obtenu par l'approche centralisée. Par contre le temps de réponse le plus petit est issu de l'approche *DIRAC* avec la réservation en mode *filling*. Les temps d'exécution sont du même ordre pour toutes les stratégies ceci s'explique par le fait que les sites possèdent en moyenne la même capacité. La différence des temps de réponse provient donc majoritairement des temps d'attente local et global. Le temps d'attente global le plus important est celui lié à l'ordonnancement *DIRAC* statique. Par contre pour cette approche le temps d'attente local comparé à l'approche centralisée est moindre. Dans le cas d'une approche *DIRAC* par réservation le temps moyen d'attente local est nul car la correspondance se fait directement à

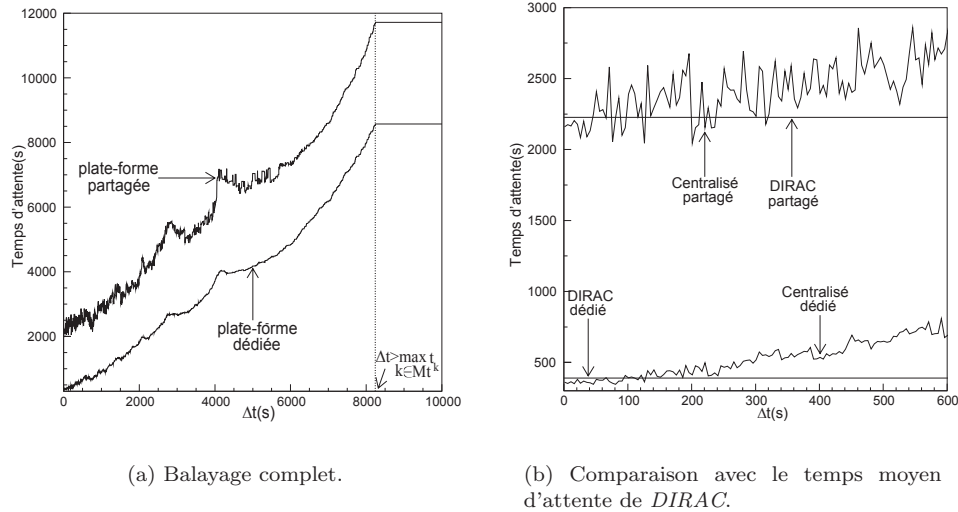


Figure 8: Temps moyen d'attente pour les méta-tâches en fonction de la période de rafraîchissement Δt du système d'information pour l'approche centralisée dans le cadre d'une plateforme dédiée et partagée.

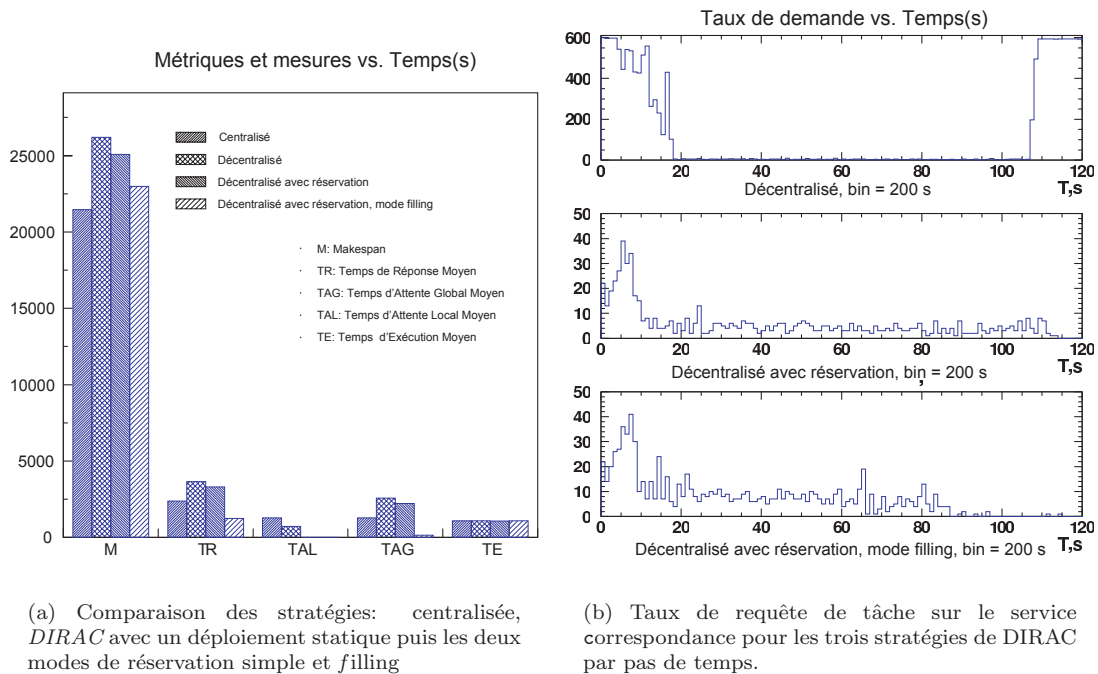


Figure 9: Résultats et caractéristiques des stratégies évaluées.

partir du nœud. Le temps d'attente s'exprime alors pour les agents soumis aux ressources. Le passage du déploiement statique à la réservation donne une amélioration de 10% sur le temps de réponse moyen. Le mode de réservation filling donne près de 50% de gain sur le temps de réponse moyen obtenu par l'approche centralisée.

Les figures 9(b) illustrent le nombre de demandes par unité de temps sur le service

correspondance, ceci dans le cas du déploiement statique (en haut), avec réservation mode simple (au milieu) et avec réservation mode *filling* (en bas). Lors de la phase d'initialisation de la plate-forme, la charge est importante sur le service correspondance pour un déploiement statique. Elle est moindre pour un déploiement dynamique. Le nombre total de demandes avec l'approche statique est de 872 et de 699 dans le cas dynamique. Nous constatons aussi que la charge sur le service correspondance est plus homogène.

8.1 Discussion

Dans un cas idéal, l'approche centralisée offre de bons résultats mais chacun sait qu'il est impossible de garantir qu'une plate-forme distribuée d'une telle envergure reste stable. Les principales et fréquentes défaillances sont les pannes réseaux, le dépassement de l'utilisation de disques, l'indisponibilité des services, les mauvaises configurations des systèmes locaux et les pannes de courant. Dans un tel contexte, il est très difficile d'avoir une image globale fiable de l'ensemble de la plate-forme. Cet ordonnancement est totalement dépendant de la performance du système d'information. Ce système souffre souvent aussi de problème d'extensibilité.

L'approche *DIRAC* contourne ce problème car une des caractéristiques de cet ordonnancement est l'absence totale de vue globale du système. Il se base sur une vue locale et partielle des ressources. Chaque ressource en fonction de son état demande et reçoit une charge adaptée à sa capacité. Les tâches sont séquentialisées et l'évènement déclencheur de l'ordonnancement et la disponibilité d'une ressource, par opposition à l'ordonnancement centralisé où l'évènement déclencheur est la soumission d'une tâche.

L'approche centralisée est très sensible en terme de performance à la moindre détérioration de la plate-forme. Cet effet est d'autant plus important si l'approche applique aussi de la prédiction. Un changement d'état impromptu d'une ressource est pris en compte après un laps de temps. Pendant ce laps de temps dans un contexte de calcul intensif, les choix sont désastreux. Ces approches souffrent souvent aussi de problème de famine pour certaines ressources en opposition à *DIRAC* où toutes ressources disponibles sont utilisées immédiatement. Avec une défaillance du système d'information l'effet de dégradation serait augmenté.

DIRAC démontre par contre des qualités naturelles d'adaptabilité. L'aspect dynamique de la plate-forme oblige un ordonnancement opportuniste, réactif et non prédictif. De plus cette approche de part les résultats obtenus est très proche en terme de performance à une approche centralisée. Elle est simple à mettre en place, plus stable et plus souple. Elle permet la réservation de ressource qui augmente considérablement ses résultats en mode *filling* au détriment des organisations concurrentes. Cette souplesse améliore les temps de réponse mais par contre fournit une charge plus régulière sur le composant correspondance. Cette amélioration a un coût qui correspond au déploiement inutile d'agents qui meurent tout de suite après la réponse négative du service correspondance (299 dans notre cas, ce qui n'est pas négligeable).

9 Conclusions et perspectives

Dans cet article nous proposons un modèle d'une plate-forme de méta-ordonnancement. Lors de la validation du simulateur découlant de cette modélisation nous avons mesuré un taux d'erreur de 12% par rapport à la réalité pour la prédiction du makespan. Avec cet outil, nous avons démontré qu'une approche centralisée est meilleure en terme de performance pour le calcul intensif qu'une approche décentralisée. Mais ceci est dans le cas idéal où la période de rafraîchissement est nulle. Au delà de 95 s dans un contexte dédié, l'approche '*Pull*' a des résultats similaires et surtout plus stable. Ceci se confirme dans un contexte partagé. L'approche '*Pull*' autorise surtout des scénarios plus riches qui permettent d'améliorer les performances de 50% par rapport à l'approche centralisée, notamment par la réservation de ressources. Il serait intéressant de mener des travaux sur la migration de tâches d'un site fortement chargé à un site moins chargé. Au sein d'une même communauté, plusieurs applications coexistent et s'exécutent de manière concurrentielle. Le critère d'ordonnancement d'une application est fortement conditionné par les

besoins et demandes de l'application elle-même. Nos prochains travaux porteront aussi sur cet aspect de méta-ordonnancement multi-critères dans un environnement partagé.

References

- [1] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, 15-18May 2001.
- [2] Doar. A better model for generating test networks. In *IEEE GLOBECOM*, 1996.
- [3] J. Closier et al. Results of the lhcb experiment data challenge 2004. In *CHEP'04*, Interlaken, November 2004.
- [4] I. Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.
- [5] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.
- [6] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.
- [7] Garonne, V. and Stokes-Rees, I. and Tsaregorodsev, A. DIRAC: A Scalable Lightweight Architecture for High Throughput Computing. In *Grid 2004, 5th IEEE/ACM International Workshop on Grid Computing*, November 2004.
- [8] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 191–202. Springer-Verlag, 2000.
- [9] L. He, S. A. Jarvis, D. P. Spooner, and G. R. Nudd. Optimising static workload allocation in multiclusters. In *IPDPS*, 2004.
- [10] L. Kleinrock, editor. *Queueing Systems Volume I : Theory*. John Wiley and Sons, 1975.
- [11] H. Li, D. Groep, and L. Walters. Workload characteristics of a multi-cluster supercomputer. In *Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2004.
- [12] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.
- [13] D. Lu and P. Dinda. Synthesizing realistic computational grids. In *Proceedings of ACM/IEEE SC 2003*, 2003.
- [14] SETI@Home. <http://setiathome.ssl.berkeley.edu/>.
- [15] G. Shao, F. Berman, and R. Wolski. Using effective network views to promote distributed application performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.
- [16] A. Takefusa. Bricks: A performance evaluation system for scheduling algorithms on the grids. In *JSPS Workshop on Applied Information Technology for Science*, 2001.
- [17] Atsuko Takefusa, Satoshi Matsuoka, Henri Casanova, and Francine Berman. A study of deadline scheduling for client-server systems on the computational grid. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, page 406. IEEE Computer Society, 2001.
- [18] S. Vadhiyar and J. Dongarra. A metascheduler for the grid. In *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, 2002.