RENPAR'16 / CFSE'4 / SympAAA'2005 / Journées Composants
Le Croisic, France, 5 au 8 avril 2005

Le HTM : un module de prédiction de performances non-intrusif pour l'ordonnancement de tâches sur plate-forme de métacomputing

Yves Caniou, Emmanuel Jeannot

LIP, ENS-Lyon
France
`Yves.Caniou@ens-lyon.fr`

LORIA, Université H. Poincaré – Nancy 1
France
`Emmanuel.Jeannot@loria.fr`

Résumé
Cet article présente un module de prédiction de performances non-intrusif que nous avons conçu et implanté dans l'intergiciel de grille NetSolve. Il tient compte des interférences que les tâches subissent lorsqu'elles sont exécutées en temps-partagé et est capable de donner la date de terminaison de *chaque tâche* dans le système. Nous présentons aussi quatre heuristiques d'ordonnancement dynamique. De nombreuses expériences réelles ont été conduites. Nos heuristiques surpassent MCT sur plusieurs critères à la fois, dont le makespan et le temps de réponse moyen.

Mots-clés : Prédiction de performances, modèle temps partagé, ordonnancement dynamique, métacomputing, heuristiques multi-critères

1. Introduction

Les applications scientifiques requièrent de plus en plus de calculs. L'augmentation de leurs besoins est bien plus rapide que celle, due aux progrès technologiques, de la puissance calculatoire des nouvelles machines. En effet, les applications fondées sur des simulations demandent plus d'itérations ou instancient des modèles plus complexes afin d'affiner leurs estimations (comme c'est le cas pour les estimations météorologiques par exemple[1]). Les domaines de la biologie et de la physique nécessitent le traitement d'un volume de plus en plus important de données[2] ou demandent simplement le maximum de ressources possible[3], voire plus généralement les deux avec l'imagerie médicale ou le projet LCG[4] de la grille de calcul du Grand Collisionneur de Hadrons (LHC)[5]. Une solution pour traiter les grands problèmes scientifiques est d'agréger les puissances de calcul des machines disponibles sur le réseau. L'agrégation de ressources hétérogènes et distribuées forme une "*grille de calcul*".
Le concept de la grille est expliqué par I. Foster et C. Kesselman dans [4]. Il s'agit d'interconnecter l'ensemble de toutes les ressources disponibles et de faire profiter l'ensemble des utilisateurs des capacités de calcul et de stockage de l'agrégat ainsi constitué. Toutefois, les ressources sont hétérogènes et distribuées géographiquement et la difficulté de leur gestion ne doit pas incomber à l'utilisateur.
L'abstraction des ressources et de leurs caractéristiques (par exemple d'ordre matériel, leur puissance ou encore leur dynamicité) est appelée le **grid computing**. Dans cet article, nous en discernons la partie **métacomputing** qui est plus proche des besoins applicatifs des utilisateurs et porte plus particulièrement sur *l'utilisation transparente des ressources de calcul*.

[1] `http://www.eurogrid.org/wp2.html`
[2] `www.griphyn.org/`, `http://www.ppdg.net/` et `http://eu-datagrid.web.cern.ch/eu-datagrid/`
[3] `http://setiathome.ssl.berkeley.edu/`
[4] `http://lcg.web.cern.ch/LCG/`
[5] `http://public.web.cern.ch/public/about/future/future.html`

L'exécution d'une application sur la grille se fait au moyen de couches logicielles intermédiaires. Parmi ces couches, nous trouvons les intergiciels de grille qui s'appuient sur des services pour distribuer l'exécution de l'application. L'efficacité dégagée dépend toutefois du déploiement des composantes de l'intergiciel ainsi que de la performance offerte à chaque application qui lui est soumise. C'est au service d'ordonnancement, qui fait l'objet de ce travail, qu'incombe l'optimisation de l'utilisation des ressources. Il s'agit de déterminer les ressources à utiliser et quand elles le seront afin d'optimiser la métrique souhaitée. Dans ce but, les stratégies d'ordonnancement nécessitent des prédictions de performances de qualité : par exemple des prédictions sur l'évolution de l'utilisation des ressources ou sur la durée des tâches.

Dans ce travail, nous considérons plus particulièrement le contexte GridRPC du modèle client–agent–serveur sur la grille que les NES (*Network Enable Server*) instancient. Le module d'ordonnancement appelé par ce type d'intergiciel prend ses décisions grâce à un algorithme dynamique. Généralement, c'est une variante de MCT (*Minimum Completion Time*) [5] qui est utilisée. Son but est de minimiser le makespan, c'est–à–dire la date de terminaison des requêtes.

Parmi les contributions de ce travail, nous proposons un module de prédiction de performances, le gestionnaire de l'historique des tâches (HTM, *Historical Trace Manager*), capable de donner une estimation fiable de la durée des tâches. De nombreuses études existent sur la prédiction de la durée des tâches. A notre connaissance, aucune n'utilise les connaissances disponibles en couplant les décisions établies au fur et à mesure par le module d'ordonnancement avec les prédictions, pour des tâches capables de s'exécuter concurremment sur les serveurs de calcul. Le HTM est donc fondé sur les interférences que des tâches concurrentes pour une ressource de calcul ont les unes sur les autres et donne ainsi le retard occasionné pour chacune d'elles.

Nous avons conçu des heuristiques d'ordonnancement dynamiques et multi–critères. Le makespan étant la métrique la plus couramment utilisée, les heuristiques tentent de plus, avec l'aide des informations délivrées par le HTM, de donner une meilleure qualité de service à chaque requête et par extension à chaque client, comme de réduire le temps de réponse moyen des requêtes.

Nous avons implanté le HTM et les heuristiques dans NetSolve [3] afin de réaliser des expériences réelles. Elles nous ont permis d'étudier concrètement l'emploi du HTM dans un environnement de résolution de problèmes, de travailler la précision de ses informations en introduisant des mécanismes de synchronisation et d'étudier l'efficacité réelle de nos heuristiques.

Cet article est organisé de la façon suivante : nous abordons les modèles utilisés dans la section 2 ; la section 3 concerne la prédiction de performances. Nous y présentons en particulier le HTM. Nous étudions son acuité dans la section 4 ainsi que la validité du modèle d'interférence. Nous décrivons nos heuristiques d'ordonnancement dynamiques, reposant sur les informations que délivre le HTM dans la section 5. Les critères d'évaluation et métriques considérées sont présentés à la section 6. Nous détaillons les expériences et les résultats obtenus à la section 7 pour finalement conclure à la section 8.

2. Modèles

2.1. Le modèle client-agent-serveur

Le modèle GridRPC [6] est un standard proposé par le GGF[6]. Ce modèle est illustré par la figure 1 et fonctionne de la façon suivante : dans la phase initialisation de l'environnement, chaque serveur déclare ses services auprès de l'agent (appelé aussi *Registry* ou encore *RMS, Resource Management System*). Lorsqu'un utilisateur requiert un service, il soumet sa requête à l'agent du système. Celui-ci lui communique en retour un identifiant donnant accès à l'application recherchée. Grâce à cet identifiant, le client contacte le serveur qui lui retournera les résultats du service exécuté.

Cette approche définit aussi le comportement de certains appels de fonction et leur contrôle, l'existence des modes synchrone et asynchrone, les retours d'erreur. En revanche, la manipulation des données comme *la persistance*, c'est–à–dire la capacité de laisser des résultats sur un serveur, et leur typage sont toujours en étude.

2.1.1. Modèle d'exécution des tâches

Dans ce qui suit nous supposerons que l'ensemble des ressources est dédié à l'environnement GridRPC d'exécution des tâches.

[6] http://www.globalgridforum.org/

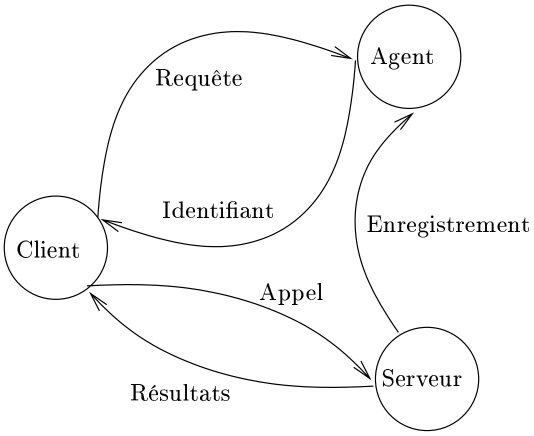

FIG. 1 – Modèle de base GridRPC

Dans le modèle client-agent-serveur, une tâche est exécutée dès son arrivée sur le serveur : il s'agit du modèle temps partagé. Dans les systèmes temps–partagé, chaque tâche doit progresser dans son exécution. Pour les variantes du système UNIX, chaque processus est exécuté pendant un tronçon de temps très bref donnant l'illusion qu'ils sont exécutés en parallèle, ce que l'on nomme *le multitâche*.
Le modèle d'exécution des tâches est donc construit autour du macro modèle suivant : s'il y a n tâches se partageant la ressource à un instant, chacune d'elles profite de $1/n$ de la puissance crête de la ressource. Ainsi, nous avons représenté en haut de la figure 2 le diagramme de Gantt illustrant l'état de la ressource au temps de soumission de la nouvelle tâche (c'est–à–dire à $t = a_3$, date d'arrivée de la tâche 3). On voit les conséquences de l'affectation de cette nouvelle tâche sur le diagramme présenté en bas de la figure. Les trois tâches se partagent la ressource uniformément jusqu'à la terminaison de l'une d'elles (ici jusqu'à T_1, date à laquelle termine la tâche 1), les autres disposant de 50% de la ressource ensuite jusqu'à la date T_2 après laquelle la tâche 3 dispose de la totalité de la puissance de la machine.

3. Prédiction de la durée des tâches dans les environnements agent-client-serveur

3.1. Problêmatique

Les ordonnanceurs des environnements de calcul GridRPC font habituellement appel à des modules de prédiction de performances reposant sur les informations délivrées par des senseurs installés sur les différentes ressources de la plate–forme. Ces informations sont généralement injectées dans des formules probabilistes ou de calculs en séries statistiques afin d'en tirer la date de terminaison de la requête nouvellement reçue comme dans NWS [8]. L'avantage de l'approche par senseur est qu'elle fonctionne même si les ressources ne sont pas dédiées à l'environnement. En revanche les senseurs, pour mesurer les performances actuelles des ressources, doivent faire des mesures intrusives et le délais de transmissions des mesures peut parfois être trop long par rapport au temps disponible pour prendre une décision.
Dans [7], Weissman a observé *le délai* à l'aide d'expériences de simulation, c'est–à–dire la différence entre la date de terminaison lors de la soumission et la date de terminaison réelle. Il déduit l'importance de la perturbation des tâches les unes sur les autres lors de la prise de décision de l'ordonnanceur.
L'approche que nous proposons ici permet de déterminer la date de terminaison d'une tâche dans le cas de ressources dédiées. Elle est fondée sur une simulation de l'ensemble des ressources et la détermination de la perturbation des tâches lors de leurs exécutions simultanées. L'avantage de cette approche par rapport à celle qui utilise des senseurs est qu'elle n'est pas intrusive et que l'information cherchée est immédiatement disponible.

3.2. Gestionnaire de l'historique des tâches

Le HTM est un module de prédiction de la durée des tâches soumises à l'environnement GridRPC. Il permet de déduire la charge interne de la plate–forme de façon non-intrusive et donne immédiatement des

réponses sur son état. Comme l'ordonnanceur, il est directement intégré au fonctionnement de l'agent.
Le HTM doit enregistrer les informations relatives à chaque requête comme sa durée et l'affectation proposée par le module d'ordonnancement. Il permet de prévoir *a priori* les prochaines variations de la charge CPU d'un serveur occasionnées par les décisions prises dans l'intergiciel. Le HTM tient compte dans ses prédictions, des perturbations résultantes de l'affectation d'une nouvelle tâche sur un serveur.
Grâce aux informations qu'il a à sa disposition, le HTM doit rendre possible une estimation de la date de terminaison *de chaque tâche* dans le système à n'importe quel instant. Il doit donc prendre en compte la perturbation que les tâches ont les unes sur les autres dans un environnement temps–partagé. L'exécution d'une nouvelle tâche pouvant dramatiquement perturber le système et remettre en cause les précédentes affectations, le module d'ordonnancement doit avoir accès à ces informations. L'heuristique utilisée peut ainsi optimiser son choix dans l'état actuel du système tout en assurant une certaine pérennité de ses décisions dans un futur proche.

3.3. Notations

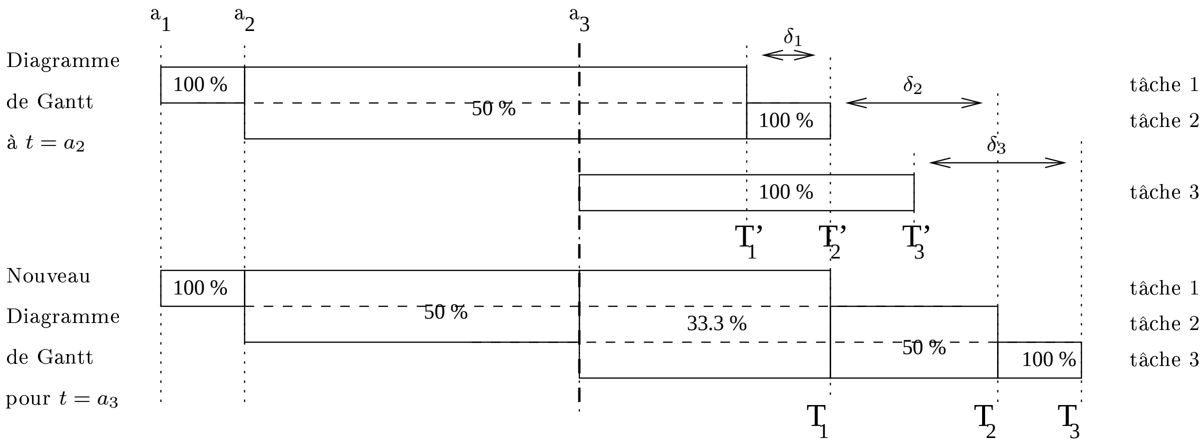

FIG. 2 – Notations du gestionnaire de l'historique des tâches, le HTM

Nous utiliserons dans tout ce qui suit les notations suivantes. Elles correspondent à ce que l'on peut observer dans la figure 2.
a_i est l'arrivée de la tâche i dans le système. T'_i est sa date de terminaison obtenue par simulation dans l'état courant du système et C_i est la date réelle de terminaison, c'est–à-dire celle observée *post-mortem* (après la fin de l'expérience). Le HTM peut simuler l'exécution d'une nouvelle tâche n et donner les nouvelles dates de terminaison T_i de toutes les tâches $i \leq n$ affectées à la ressource. On définit pour tout $k \leq n$, la perturbation $\delta_k = T_k - T'_k$ occasionnée par l'affectation de la nouvelle tâche sur chaque tâche en train d'être exécutée sur la ressource. Par extension, on dira par la suite que δ_k est la perturbation *engendrée par la tâche* n. De plus, on dira que le temps restant de la tâche k est la valeur $D_k = T_k - a_n$, pour tout $k \leq n$; cette valeur est généralement différente du temps restant de la tâche sur la ressource à vide. Enfin, on notera $P(i)$ la ressource où la tâche i est affectée et d_i sa durée totale sur la ressource à vide.

4. Validité du modèle d'interférence et précision du HTM

Dans cette section nous comparons la durée réelle d'exécution des tâches avec l'estimation faite par le HTM. Nous poursuivons deux objectifs : premièrement, déterminer la validité du modèle temps-partagé par rapport aux interférences entre les tâches et deuxièmement, évaluer la précision du HTM.

Tâche	Date d'arrivé	Taille de la matrice	Date de terminaison réelle	Date de terminaison simulée	différence	Pourcentage d'erreur
1	33.00	1500	80.79	79.99	0.8	1.7
2	59.92	1200	92.08	93.19	-1.11	3.4
3	73.92	1800	142.79	142.50	0.29	0.4
1	29.41	1500	76.69	76.29	0.4	0.8
2	56.43	1200	89.15	89.50	-0.35	1
4	96.41	1200	136.97	139.40	-2.43	5.9
6	140.41	1200	204.84	204.85	-0.01	0.02
3	70.42	1800	210.61	195.74	14.87	10.6
5	121.43	1500	235.38	232.92	2.46	2.2
8	181.45	1200	248.02	248.56	-0.54	0.8
9	206.41	1200	259.91	261.63	-1.72	3.2
7	166.42	1800	289.08	288.91	0.17	0.1

TAB. 1 – Deux expériences sur le modèle temps partagé

4.1. Évaluation du modèle temps-partagé

Nous avons tout d'abord expérimenté le modèle temps partagé sur des systèmes LINUX et Solaris quand des tâches sont des multiplications de matrices et ont la même priorité. Dans le tableau 1 nous présentons deux expériences où les tâches sont triées par date de terminaison. Le pourcentage d'erreur est défini comme 100 fois la valeur absolue de la différence divisée par la durée réelle de la tâche. En moyenne cette erreur est très faible puisqu'elle est inférieure à 3%. On peut constater que, durant ce test, au plus cinq tâches sont exécutées en même temps (tâches 3, 4, 6, 7 et 8 entre les dates 181,45 et 204,84). Dans ce cas, la tâche 3 est la plus impactée puisque toute les autres tâche ont interféré avec celle-ci (elle commence alors que la tâche 1 n'est pas terminée et finit après le début de la tâche 9).

4.2. Précision du HTM sans synchronisation

Dans cette section nous analysons la précision des estimations du HTM alors que celui-ci n'est jamais synchronisé avec l'environnement.

Pour évaluer le HTM, la précision de ses estimations a été enregistrée pendant des expériences de soumissions de tâches sur une plate-forme réelle. Pour cela nous avons implanté le HTM dans NetSolve [3].

Pour étudier la précision, nous considérons l'erreur relative entre ce qui est mesuré et ce qui est estimé. Les résultats sont montrés dans les figures 3 et 4. Les graphes se lisent comme suit. L'axe des abscisses représente la date de soumission de chaque tâche, et deux informations apparaissent selon l'axe des ordonnées. En grid foncé apparaît le ratio de la dernière estimation de la durée faite par le HTM par la durée réelle constatée à la fin du run (qu'on appelle *durée post-mortem*). Ce ratio est la dernière estimation faite par le HTM par la division de la durée réelle. L'estimation est donc celle faite lors de la soumission de la tâche et mise à jour en fonction des soumissions ultérieures. Plus le ratio est proche de la valeur 1, plus la précision est bonne. Lorqu'il est supérieur à un, l'estimation de la durée est plus grande que celle observée en réalité. En gris clair, une deuxième information apparaît sur les graphes. Il s'agit du nombre réel constaté de tâches ayant été exécutées concurremment à la tâche soumise à la date donnée.

Nous avons mené plusieurs expériences de soumission des tâches et nous avons relevé trois grands types de comportements dans l'évolution des estimations du HTM.

La partie gauche de la figure 3 montre la précision obtenue pour une expérience où des tâches indépendantes arrivent à une fréquence moyenne de 20 secondes et sont réparties sur différentes ressources. On constate qu'à de rares exceptions, les durées estimées des tâches sont très proches mais restent supérieures aux durées réelles. Là où les ratios augmentent, on observe qu'il y a au moins quatre interférences avec des tâches différentes. L'erreur moyenne de l'estimation faite par le HTM est de 3.1 % pour un écart–type 2.4.

Dans la partie droite de la figure 3 des tâches indépendantes sont soumises avec une fréquence moyenne de 17 secondes. On remarque que le HTM surestime encore les durées des tâches (les ratios sont supérieurs

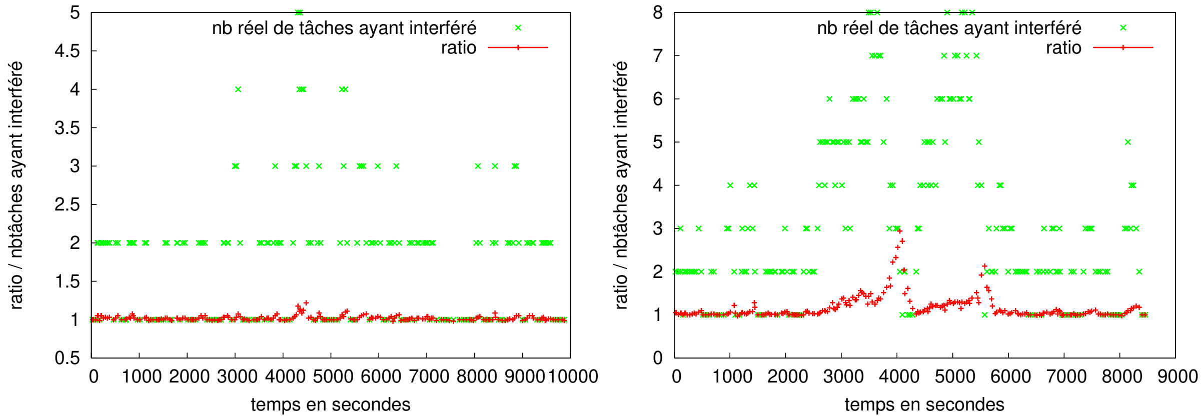

FIG. 3 – Gauche : la précision du HTM ne se dégrade pas au cours du temps. Droite : la précision du HTM se dégrade passagèrement au cours du temps

à 1), on s'aperçoit aussi que l'évolution n'est pas constante au cours du temps. Comme il y a beaucoup d'exécutions concurrentes sur le serveur considéré (plusieurs pics à 8 tâches qui interfèrent pendant l'exécution de celle observée), on observe un accroissement des erreurs conséquent sur les estimations des durées des tâches suivantes. On constate ainsi un ratio de $2,84$ pour $t = 3905$ secondes ! L'erreur moyenne constatée est de 11.7 % avec un écart–type de 13.4. En revanche, le HTM regagne complètement son acuité peu de temps après ce pic : il est ici capable de s'autoréguler ce qui lui donne une certaine robustesse.

Nous présentons dans la partie gauche de la figure 4 les résultats que nous avons observés lors de l'exécution d'un *graphe de tâches* par NetSolve. L'échelle des ordonnées est logarithmique en base 2. On constate que la qualité des estimations se dégrade très rapidement. Elle décroît toujours avec le nombre de tâches qui interfèrent avec une tâche donnée, mais comme la figure le montre, la dégradation semble plus persistante puisque pour $t > 5000$ secondes, on observe une baisse du nombre d'interférences et la précision des estimations ne revient qu'à la fin du run. Comme nous soumettons un graphe de tâches, la date de début d'une tâche dépend de la fin de ses prédécesseurs. Les petites erreurs du HTM dues au trop grand nombre de soumissions se cumulent jusqu'à ce que celui-ci se désynchronise complètement.

4.3. Précision du HTM avec synchronisation

Dans cette partie nous présentons une amélioration du HTM. Il s'agit de resynchroniser le HTM chaque fois qu'une tâche termine. Pour cela, nous avons dû encore modifier NetSolve. Chaque fois qu'un serveur finit d'exécuter une tâche, il prévient l'agent. Le HTM peut alors mettre à jour définitivement la date de terminaison de cette tâche et calculer les interférences en fonction de cette information.

Les résultats obtenus sur la précision révèlent une nette amélioration des prédictions. La partie droite de la figure 4 montre l'influence du nombre de tâches ayant interféré dans l'acuité des prédictions du HTM (l'échelle des ordonnées est logarithmique en base 2). Il s'agit de la même expérience que celle de la partie gauche de la figure 4. Nous pouvons remarquer que passé un nombre d'interférences compris entre 10 et 12 la précision se dégrade un peu. Les erreurs commencent alors à devenir importantes mais dès que le nombre d'interférences diminue et redevient inférieur au seuil, l'erreur redescend en dessous de 5%. Au total, l'erreur moyenne relevée est seulement de 5.5% avec un écart–type de 5.

4.4. Conclusion

Nous voyons que la précision du modèle d'interférence dépend du nombre de tâches qui s'exécutent en même temps. La table 1 et la figure 3 montrent que lorsque le nombre de tâches simultanées est plus petit que 5, le modèle prédit ce qui se passe : chacune des n tâches qui s'exécutent reçoit $1/n$ de la puissance CPU.

Quand plus de 5 tâches s'exécutent concurremment, le modèle tend à sur-estimer la durée de celles-ci. En conséquence la précision du modèle se dégrade. Pour des tâches indépendantes, le HTM est capable de

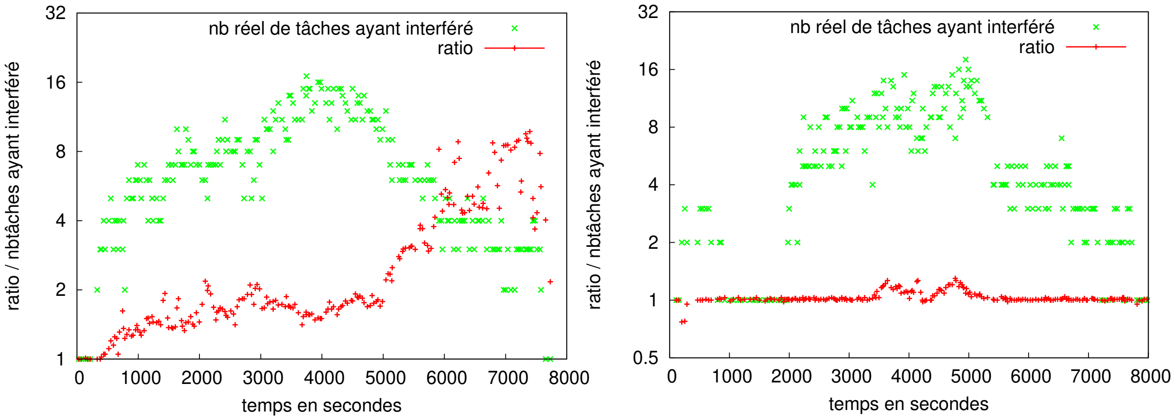

FIG. 4 – Lorsque l'on soumet des graphes de tâches, la précision du HTM se dégrade (gauche). Si on synchronise le HTM avec les erreurs ne se cumulent plus (droite)

retrouver et de conserver une bonne précision. Pour certain graphes de tâches et certaines heuristiques spécifiques, l'erreur est cumulative. Il est nécessaire, dans ce cas, de resynchroniser le HTM avec la réalité.

5. Heuristiques d'ordonnancement

Nous proposons ici plusieurs heuristiques d'ordonnancement dans le cas où le HTM est synchronisé avec les serveurs.
Toutes les heuristiques que nous avons construites et testées reposent sur les informations que leur propose le HTM. Ses estimations sont obtenues par simulation de la nouvelle requête sur la modélisation de la plate–forme. Cette évaluation prend donc en compte les délais occasionnés par l'exécution concurrente des tâches sur les ressources.
Le but de ces heuristiques d'ordonnancement est d'améliorer les performances de celles qui sont habituellement utilisées dans les environnements de résolution de problèmes, notamment MCT [5].

5.1. Historical MCT : HMCT

Inspirée de la stratégie de placement de MCT, HMCT alloue la requête au serveur sur lequel la simulation indique qu'elle termine le plus tôt (Figure 5, gauche) : on espère ainsi minimiser la date de terminaison de l'application, c'est–à–dire le makespan, en minimisant le temps de terminaison de chaque nouvelle requête qui est potentiellement la dernière requête. Tout comme pour MCT, ceci suppose donc que toute requête est une composante d'une unique application et que c'est la date de terminaison de la dernière requête qui donne le makespan.

5.2. Minimum Perturbation : MP

MP choisit le serveur qui minimise la somme des perturbations occasionnées par l'affectation de la nouvelle requête sur ce serveur (Figure 5, droite). En cas d'égalité, par exemple tout au début de l'expérience lorsqu'aucun serveur n'est encore chargé, le serveur minimisant la date de terminaison de la requête est choisi (on applique donc en cas d'égalité la stratégie orientée makespan de HMCT).
L'objectif de MP est de délivrer une meilleure qualité de service à chaque tâche en cherchant à retarder le moins possible les tâches actuellement en cours.

5.3. Minimum SumFlow : MSF

MSF utilise les informations du HTM afin de calculer la somme des durées de chaque requête d'un serveur, avant et après simulation de l'affectation de la dernière requête (Figure 6, gauche). Ensuite, l'heuristique

Pour chaque nouvelle tâche t
  Pour chaque serveur j
    Le HTM simule t sur j
    $T_{t,j} \leftarrow$ date de terminaison de t
  Allouer t à j_0 tel que $T_{t,j_0} = \min_j T_{t,j}$
  Informer le HTM que t est allouée sur j_0

Pour chaque nouvelle tâche t
  Pour chaque serveur j
    Le HTM simule t sur j
    $P_j \leftarrow \sum_i \delta_i$
  Si tous les P_j sont égaux
    Allouer t à j_0 qui minimise $T_{t,j}$
  Sinon allouer t à j_0 tel que $P_{j_0} = \min_j P_j$
  Informer le HTM que t est allouée sur j_0

FIG. 5 – Gauche : heuristique HMCT. Droite : heuristique MP

Pour chaque nouvelle tâche t
  Pour chaque serveur j
    Le HTM simule t sur j
    $P_j \leftarrow \sum_i \delta_i + T_t - a_t$
  Allouer t à j_0 tel que $P_{j_0} = \min_j P_j$
  Informer le HTM que t est allouée sur j_0

Pour chaque nouvelle tâche t
  Pour chaque serveur j
    Le HTM simule t sur j
    $P_j \leftarrow \sum_i D_i$
  Allouer t à j_0 tel que $P_{j_0} = \min_j P_j$
  Informer le HTM que t est allouée sur j_0

FIG. 6 – Gauche : heuristique MSF. Droite : heuristique ML

retourne l'identité du serveur qui minimise l'ensemble des sommes calculées, plus précisément :

$$\min_j \left(\sum_{i=1}^{i=n} (T_{i,k} - a_{i,k}) \right)$$

5.4. Minium Length : ML

La stratégie utilisée par ML consiste à minimiser la quantité de travail pondérée qu'il reste à produire au serveur avant de devenir oisif. Aussi, après la simulation de la nouvelle requête sur l'ensemble des serveurs, ML choisit le serveur qui minimise la somme des durées simulées restantes des tâches en cours (Figure 6, droite).
L'objectif de ML est de réduire les délais supplémentaires que l'affectation de la nouvelle requête va nécessairement occasionner sur celles présentes sur le serveur, tout en conservant une stratégie orientée makespan.

6. Critères d'évaluation et métriques

6.1. Notations

On note a_i l'arrivée de la tâche i. Sa durée sur le serveur de calcul j non–chargé est notée $d_{i,j}$ et sa durée estimée $e_{i,j}$. La date de terminaison est notée $C_{i,j}$ et vaut $a_i + e_{i,j}$. Lorsque le serveur j est implicitement connu, on note $C_i = C_{i,j}$. Enfin, P_i est la ressource de calcul où la tâche i est affectée.

6.2. Définitions

Du côté système, **l'utilisation des ressources** est déterminante. Il faut employer les ressources le plus possible pour les rentabiliser. Elle s'apprécie par exemple en mesurant les intervalles de temps où la ressource n'a pas été utilisée (on dit qu'elle est non–chargée, vide, libre ou *idle*) ou en calculant le **sumflow**. Du côté applicatif, on parle du **temps de réponse** ou **flow** d'une tâche pour désigner sa durée une fois celle–ci terminée et du **makespan** pour son temps de terminaison.

Scénario	Application(s)		Tâches indépendantes		Expérience	
	nbclients	larg. x haut.	nbtâches	μ (sec)	nbgraines x nbrun	nbtâches
(a) 500 tâches indépendantes	-	-	500	20, 17 et 15	3 x 2	500
(b) 1D-mesh + 250 tâches indep.	5	1x50	250	20	3 x 2	500
(c) stencil (task 1)	1	10x50	-	-	1 x 1	500
(d) stencil + 86 tâches indep	1	10x25	86	40	4 x 2	424
(e) stencil + 86 tâches indep.	1	5x25	86	25	3 x 3	211
(f) 1D-mesh	2	1x50	253	25	4 x 3	353

TAB. 2 – Scenarii, modalités et nombre d'expériences

Sumflow [1].

C'est la somme des durées des tâches qui ont été exécutées sur le serveur considéré. Pour le serveur j, c'est donc la quantité

$$sumflow = \sum_{i|P_i=j} (C_{i,j} - a_i)$$

Temps de réponse.

Il permet de mesurer la réactivité du système et sa performance à assurer des délais d'exécution. On le définit pour chaque tâche i par

$$f_i = C_{i,j} - a_i$$

Makespan.

Le *makespan d'un ensemble de tâches* S (regroupées ou non dans une application) est le temps de terminaison de la dernière tâche qui finit son exécution, i.e.,

$$makespan_S = \max_i C_i$$

7. Résultats

Nous avons testé nos heuristiques dans le cas où le HTM est synchronisé avec la réalité [2]. Plusieurs scénarii, résumés dans le tableau 2 ont été executés. Le scénario (a) consiste en la soumission de tâches indépendantes, les autres en la soumission d'une ou plusieurs applications en parallèle à des tâches indépendantes (sauf scénario (c)).
Trois tailles d'un même type de tâche intensive en calcul ont été utilisées pour les générer. Une tâche demande respectivement 16, 31 et 46 secondes sur le plus rapide des serveurs dédiés. Le temps de transmission est inférieur à 10 % du temps d'exécution. Sauf pour les applications stencil qui ne sont composées que de tâches de taille 1, le choix de la taille d'une tâche s'opère selon une loi uniforme. L'inter-arrivée des tâches indépendantes suit une loi de Poisson de paramètre μ et leur nombre est suffisamment grand pour que l'expérience termine par l'une d'elles. Chaque scénario a été instancié un nombre de fois *nbgraines* avec une graine différente. Chaque instance ainsi générée a été exécutée un nombre *nbrun* de fois.
Ce que nous présentons ici représente près de 25 jours non stop de calcul.
Nous présentons les performances des heuristiques HMCT, MP, MSF et ML en comparaison aux performances que donne MCT. L'ensemble des résultats est axé sur quatre graphiques où sont respectivement donnés pour chaque heuristique et selon le scénario :
- les gains moyens sur le makespan *pour chaque application* (scénarii (b), (c), (d), (e), (f), Figure 7, gauche),
- les gains moyens réalisés sur le sumflow *pour chaque application* (Figure 7, droite),
- les gains moyens réalisés sur le sumflow pour l'ensemble des tâches indépendantes, qui sont par définition les mêmes que ceux observés sur le temps de réponse *de chaque tâche* (Figure 8, gauche),
- le pourcentage moyen de tâches indépendantes terminant plus tôt lorsque le scénario est ordonnancé par l'une de nos heuristiques par rapport à MCT (Figure 8, droite).

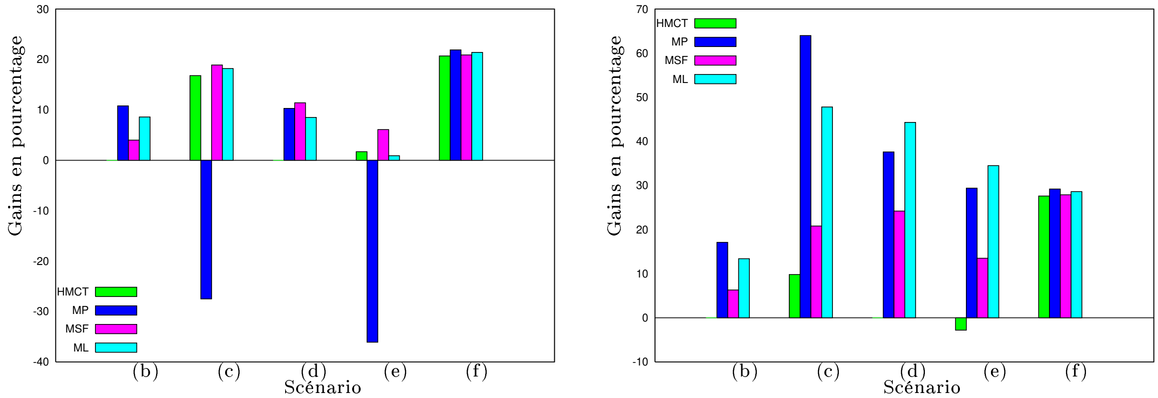

FIG. 7 – Gauche : gains moyens réalisés *par chaque client* sur le makespan de son application selon l'heuristique et le scénario. Droite : gains moyens réalisés sur le sumflow *pour chaque client* selon l'heuristique et le scénario

Remarquons que lorsque la performance d'HMCT n'apparaît pas, cela signifie que l'heuristique a surchargé le serveur le plus rapide qui a refusé du travail. L'ensemble des tâches n'ayant pas été exécuté (le refus d'une tâche d'une application bloque sa soumission), les résultats ne peuvent être validés.

Makespan.

La partie gauche de la figure 7 montre que HMCT, MSF et ML donnent toujours de meilleurs résultats que MCT sur le makespan, avec une date de terminaison de chaque application jusqu'à 22% plus tôt. En revanche on constate avec les scénarii (c) et (e) que MP peut avoir des difficultés pour ordonnancer une application à cause de l'affectation de tâches critiques sur un serveur lent. On remarque encore que la stratégie de placement de MP, MSF et ML assure la faisabilité de la totalité des expériences et que MSF et ML donnent en moyenne de meilleurs résultats que HMCT.

Sumflow.

La partie droite de la figure 7 indique les gains moyens obtenus en terme de coût total en temps pour chaque application. On constate qu'en dehors du scénario (e) où HMCT donne des performances négatives, toutes nos heuristiques réduisent le sumflow. MP atteint même une économie de 64% pour le scénario (c) et ML juste derrière avec 45%. On remarque aussi la constance des très bons résultats de ML tout au long des scénarii.

Temps de réponse moyen.

De façon générale, nos heuristiques améliorent le temps de réponse moyen (et donc le coût en temps que requiert l'exécution de l'ensemble des tâches indépendantes) que demandent les tâches ordonnancées par MCT (Figure 8, gauche). On voit que la stratégie de placement de MP paye dès que la fréquence de soumission est suffisamment élevée pour dégager de nombreuses interférences : le seul scénario où MP obtient de moins bons résultats que MCT est (a) pour $\mu = 20$ secondes. MSF et ML donnent les mêmes performances sur les scénarii (a) avec $\mu = 20$ et $\mu = 17$ secondes et (f), mais ML permet des gains supérieurs sur les autres avec jusqu'à 4 fois plus d'économie en temps de calcul sur l'environnement pour le scénario (e).

Pourcentage de tâches terminant plus tôt.

Le graphe de la partie droite de la figure 8 représente pour chacune de nos heuristiques et pour chaque scénario, le pourcentage moyen de tâches qui ont terminé plus tôt que si elles avaient été placées par MCT. Lors du calcul du pourcentage, nous avons considéré que deux tâches terminaient *à la même date* si la différence entre les valeurs étaient inférieure à deux secondes. Nous avons donc aussi représenté le pourcentage de tâches qui terminent plus tôt avec MCT. Un gain apparaît dès que la surface du rectangle

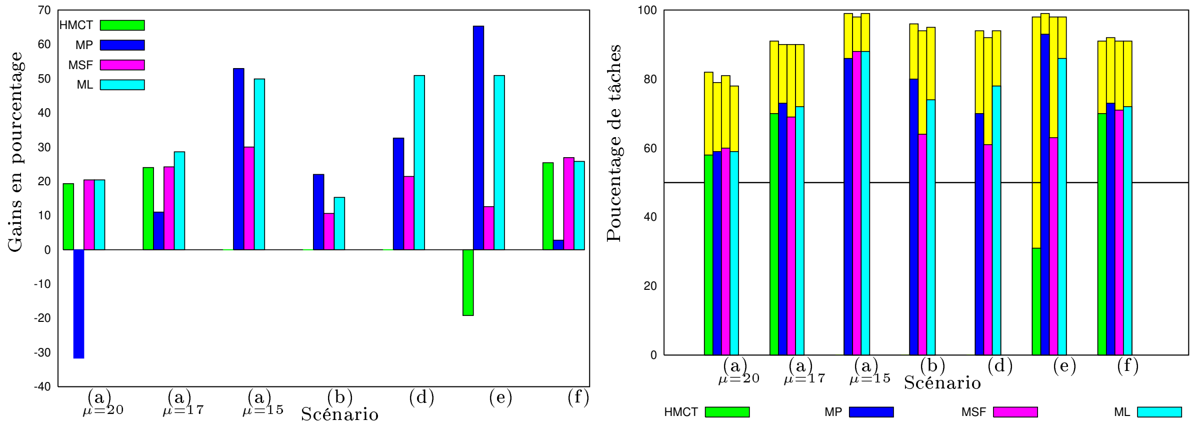

FIG. 8 – Gauche : gains moyens réalisés sur le temps de réponse *de chaque tâche* selon l'heuristique et le scénario. Droite : pourcentage de tâches terminant plus tôt et aussi tôt qu'avec MCT, selon l'heuristique et le scénario

de l'heuristique considérée est plus importante que celle du rectangle supérieur associé. Hormis pour HMCT au cours du scénario (e), on s'aperçoit qu'une tâche a plus de chance de finir tôt si l'environnement de calcul utilise l'une de nos heuristiques quel que soit le scénario considéré. MP, MSF et ML établissent les meilleurs résultats, avec par exemple le scénario (e) où MP atteint 93% et ML 86%.

8. Conclusion

Dans cet article nous avons étudié l'ordonnancement de tâches dans le modèle GridRPC. En ce qui concerne la performance, l'ordonnancement est un service majeur que doit fournir un environnement de grille de calcul.

Nous avons montré que la problématique de la prédiction de performance doit être résolue pour pouvoir mettre au point des heuristiques d'ordonnancement efficaces. Dans cette optique, nous avons proposé le gestionnaire de l'historique des tâches (HTM). Il s'agit d'un module non-intrusif (il n'utilise pas de senseur) pour prédire l'utilisation des ressources par les tâches. Nous avons implanté le HTM dans NetSolve.

Nous avons étudié la précision du HTM et montré que, lorsque l'on ordonnance des graphes de tâches, celui-ci peut se désynchroniser avec la réalité. Pour palier à ce problème, nous avons synchronisé la terminaison des tâches dans le HTM. Le HTM synchronisé montre de très bonnes performances de prédiction quel que soit le type de soumission testée.

Nous avons proposé 4 heuristiques que nous comparons avec MCT qui est implantée dans NetSolve. Nous proposons une approche multi-critères où plusieurs métriques sont évaluées en même temps. Nous montrons que parmi les heuristiques proposées, MSF et ML offrent des performances supérieures à MCT dans tous les cas de figures étudiés.

Dans le futur nous souhaitons travailler à la distribution du HTM dans des environnements de grilles multi-agents comme DIET.

Bibliographie

1. K.R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.
2. Y. Caniou. *Ordonnancement sur une plate-forme de métacomputing.* PhD thesis, Université UHP Nancy 1, France, 2004.
3. H. Casanova and J. Dongarra. Netsolve : A network server for solving computational science problems. In *Proceedings of Super-Computing -Pittsburg*, 1996.
4. I. Foster and C. Kesselman. *The Grid : Blueprint for a New Computing Infrastructure.* Morgan-Kaufmann, 1998.
5. M. Maheswaran, S. Ali, H.J. Siegel, D. Hengsen, and R.F. Freund. Dynamic matching and scheduling

of a class of independent tasks onto heterogeneous computing system. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, april 1999.

6. H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. A gridRPC model and API for end-user applications. December 2003. `https://forge.gridforum.org/projects/gridrpc-wg/document/GridRPC_EndUse%r_16dec03/en/1`.
7. J.B. Weissman. The interference paradigm for network job scheduling. In *Proceedings of the 10th International Parallel Processing Symposium, HCW*, 1996.
8. R. Wolski, N.T. Spring, and J. Hayes. The network service : A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6) :757–768, october 1999.