RenPar'20 / SympA'14 / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

Sensibilité des Algorithmes d'Ordonnancement

Jean-Noël Quintin, Swann Perarnau
{quintin,perarnau}@imag.fr

Équipe-Projet MOAIS, LIG/INRIA, Grenoble

Résumé
Cet article étudie comment la performance de divers algorithmes d'ordonnancement varie lorsqu'ils sont confrontés à des conditions non prises en compte dans le cadre théorique ayant servi à leur conception.
On peut par exemple imaginer que la plateforme sur laquelle est déployé un algorithme soit utilisée pour des applications différentes de celles initialement envisagées. Dans certains cas, l'algorithme peut s'avérer inadapté : générant trop de communications ou réalisant de mauvais choix d'ordonnancement. Il est donc primordial d'étudier si la performance des ordonnanceurs se dégrade (et dans quelles proportions) lorsque leur environnement est modifié. Nous montrons en effet sur des simulations comment des modifications *fines* des caractéristiques des graphes en entrées peuvent impacter la performance d'algorithmes de la littérature. Nous observons notamment comment une modification de la distribution des coûts en calculs présents sur les graphes de tâches peut inverser une comparaison entre deux algorithmes.

Mots-clés : ordonnancement, simulation, performance

---

1. Introduction

Les machines du Top500 [7] sont de plus en plus complexes. Elles possèdent plusieurs dizaines de milliers de processeurs, interconnectés par différents réseaux haute performance constituant une hiérarchie difficile à appréhender. Les applications déployées sur ces machines changent aussi. Le Graph500 [8], par exemple, illustre ces changements à travers l'utilisation d'applications bien plus coûteuses en entrées/sorties que celles utilisées jusqu'alors.
Pour atteindre une performance acceptable, les ordonnanceurs présents sur ces machines se doivent donc de prendre en compte le coût des communications engendrées par leurs décisions mais aussi d'être *résistants* à ces nouveaux usages. Malheureusement, les algorithmes d'ordonnancement les plus classiques de la littérature n'ont pas été étudiés théoriquement sur ce type de modifications de l'environnement.
Nous nous intéressons ici aux conséquences que peuvent avoir ces modifications : comment se comporte la performance d'un algorithme lorsqu'il est mis en pratique dans un cadre différent de celui pour lequel il a été conçu et validé ?
Il s'agit là de la contribution majeure de cet article : nous étudions la **sensibilité** d'un ordonnanceur à l'un des paramètres de son environnement (plus précisément aux caractéristiques des applications ordonnancées).
Pour cela, nous avons simulé un ensemble d'algorithmes d'ordonnancement bien connus (et prenant en compte les communications). Ces simulations suivent toutes le même schéma : après avoir étudié la performance d'un algorithme sur un *groupe témoin* de graphes de tâches, nous observons comment cette dernière varie lorsque l'on modifie une des caractéristiques du graphe.
La suite de l'article est organisée de la façon suivante : la prochaine section rappelle quelques définitions de base de l'ordonnancement. La section 3 détaille les différents outils que nous avons utilisés pour réaliser notre étude ainsi que sa mise en œuvre. La section 4 présente les résultats obtenus lors de cette campagne expérimentale. Une des conséquences les plus intéressantes de cette étude est l'apparition

d'inversions : un algorithme A qui semblait plus performant qu'un autre algorithme B sur un jeu de données devient moins efficace sur un autre jeu de données proche du premier. Nous concluons cet article en section 5.

2. Définitions

Un ordonnanceur est un algorithme allouant à un ensemble de tâches des ressources (processeurs, machines, ...), avec pour objectif d'optimiser un ou plusieurs critères de performance. Une instance du problème est caractérisée par un graphe orienté acyclique $G = (V, E)$, par un entier m représentant le nombre de ressources disponibles, par un temps d'exécution p_i de la tâche i et une quantité de données à échanger c_{ij} entre les tâches i et j (avec $i, j \in V$). Les arêtes E représentent à la fois une contrainte de précédence et l'existence d'une communication entre deux tâches : si $(i, j) \in E$ alors la tâche j ne peut être exécutée avant la terminaison de la tâche i et du transfert des données associées.
Le coût des communications (le temps nécessaire à leur complétion) est lié à la quantité de données échangées mais aussi aux caractéristiques de la plateforme d'exécution. En effet, le couple de machines impliquées dans la communication influence le temps de transfert. Par exemple, on considère que le temps d'une communication entre deux tâches sur la même machine est nul.
Dans la notation à trois champs [5], nous nous intéressons ici au problème $P|prec, p_i, c_{ij}|C_{max}$. Il correspond à un graphe de tâches avec précédence et coût de communications s'exécutant sur m processeurs homogènes. Le critère de performance est le temps de complétion de la dernière tâche, qu'il faut donc minimiser. En terme de complexité, ce problème est NP_{hard} et inapproximable [6].
La prise en compte des communications représente une des difficultés majeures de ce problème, l'analyse théorique étant en effet dépendante du modèle de communication utilisé.
Le modèle le plus simple pour cette prise en compte des communications est le modèle délai. Dans ce cas, le temps de communication entre deux tâches croit linéairement avec la quantité de données échangées. Durant cette communication, les différentes machines peuvent continuer l'exécution d'autres tâches librement. Ce modèle est peu réaliste : il ne prend pas en compte la contention sur le réseau ni la perturbation subie par les machines impliquées (émettrice et réceptrice). D'autres modèles existent, permettant de modéliser plus finement le comportement du réseau, comme par exemple BSP [13] et LOGP [3]. Il sont néanmoins bien moins utilisés dans des analyses théoriques car ils les complexifient fortement.

3. Simulation d'ordonnanceurs

Afin d'étudier la sensibilité de différents algorithmes à certains paramètres des graphes de tâches, nous avons réalisé une campagne de simulations à l'aide de Grid'5000. Cette section décrit les différents algorithmes choisis, les outils utilisés ainsi que le principe général des expériences réalisées.

3.1. Algorithmes étudiés

Dans la littérature, de nombreux algorithmes d'ordonnancement prenant en compte les communications existent. Dans la suite, nous nous intéressons plus particulièrement à deux catégories, les algorithmes de *list-scheduling* et de *clustering*.
Parmi les algorithmes de *list-scheduling*, nous comparons les algorithmes HEFT [12], CPOP [12] et HBMCT [9].
Les algorithmes de *list-scheduling* fonctionnent tous selon le même principe : ils placent les tâches prêtes à être exécutées dans une liste à priorité et, de façon gloutonne, attribuent ces tâches aux processeurs libres. La priorité peut dépendre de différents facteurs (le coût des communications, la taille des tâches, ...). À ce titre, HEFT définit la priorité d'une tâche comme la longueur du chemin le plus long partant de celle-ci (en prenant en compte les temps d'exécution et les temps de communication).
Pour les algorithmes de *clustering*, le fonctionnement est différent. Le but de ces algorithmes est de regrouper les tâches qui génèrent des communications impactant le temps d'exécution total sur une infinité de machines. Pour cela, ils regroupent de manière itérative sur les mêmes processeurs les tâches concernées par les communications les plus importantes. Les groupes formés, appelés *clusters*, sont ainsi fusionnés, à la condition que le chemin critique n'augmente jamais lors d'un regroupement. Si ce regroupement est toujours réalisé à partir d'un ordonnancement sur une infinité de machines, il n'atteint pas

nécessairement un nombre de clusters inférieur au nombre de processeurs disponibles dans le problème considéré. La plupart des algorithmes considèrent alors que le problème ne peut être résolu.
Nous avons choisi comme algorithme de *clustering* DSC [14] qui fourni un ordonnancement quelque soit le nombre de processeurs, et c'est l'implémentation de PYRROS [10] que nous avons utilisée pour nos simulations.

3.2. Outils utilisés

Afin de réaliser notre étude dans les meilleures conditions possibles nous avons fait appel à deux outils : Simgrid [1] et GGen [2].
GGen est un outil de génération de DAG dédié à l'ordonnancement. Il propose une implémentation de qualité de diverses méthodes de génération souvent utilisées en ordonnancement. Cet outil repose sur un format de données très simple (DOT [4]) et permet d'annoter les graphes (nœuds comme arêtes) à l'aide de différentes distributions classiques (gaussienne, uniforme, Pareto, ...). Cette séparation entre la génération de la structure du graphe et son annotation nous a notamment permis dans notre étude de conserver systématiquement les mêmes dépendances entre les tâches tout en faisant varier les coûts en calcul ou en communication présents sur le graphe.
Simgrid est un outil permettant la réalisation d'un simulateur gérant finement les coûts des communications. Pour modéliser ces derniers, Simgrid prend en entrée une description de la plateforme, fournissant les caractéristiques des liens de communication et la puissance des machines. Il permet de simuler la contention réseau et l'impact des communications sur le travail réalisé par les machines communicantes. Nous y avons ajouté une interface permettant l'interprétation des DAGs générés par GGen comme graphes de tâches. Ainsi en fournissant la plateforme et le graphe de tâche, une trace du détail de l'exécution est fournie.

3.3. Expériences réalisées

L'objectif de nos simulations est d'établir si un des algorithmes précédemment cités est sensible à certaines caractéristiques des applications. Pour cela, nous allons comparer la performance des algorithmes selon que l'on utilise un jeu d'entrées *test* (ou témoin) ou une version modifiée de ce dernier. Deux types d'entrées vont être modifiées par la suite : la distribution des coûts en calcul et celle des coûts en communication des graphes de tâches considérés par les ordonnanceurs.
La plateforme d'exécution de référence est un groupe de machines homogènes reliées entre elles par un lien privilégié (pas de contention sur les communications). Cette plateforme nommée *clique* dispose d'un réseau gigabit. Les caractéristiques (latence, débit) du réseau ne seront pas modifiées par la suite. Nous appelons dans les paragraphes suivants temps de communications le temps moyen que prennent les transferts de données lorsqu'ils sont simulés sur cette plateforme.
La caractéristique principale qui a motivé notre choix de graphes de tâches de référence est le ratio calculs/communications. En effet, ce ratio a une influence capitale sur la capacité d'un ordonnanceur à distribuer les tâches sur les différentes machines disponibles. Nous disposons donc de trois jeux d'entrées servant de référence : ils partagent les mêmes caractéristiques en nombre de nœuds ou d'arêtes, mais contiennent des coûts en calcul et en communication différents. Le tableau 1 récapitule leurs caractéristiques ainsi que le ratio temps moyen de calcul d'une tâche sur temps moyen de communication (CCR).
Les graphes ont été générés à l'aide de l'algorithme Layer-By-Layer [11] : l'ensemble des nœuds est réparti entre 100 niveaux, puis chaque arête possible est ajoutée au graphe avec une certaine probabilité. Deux nœuds appartenant au même niveau ne peuvent avoir d'arêtes entre eux. La probabilité d'une arête a été choisie de façon à obtenir une moyenne de 3 arêtes (entrantes et sortantes) par nœuds [1]. Chaque groupe témoin contient 100 graphes.
La section suivante présente donc un ensemble de simulations réalisées, d'abord sur les jeux de données témoins puis sur des modifications de ces jeux de données.

1. la formule exacte est : $p = \frac{3l}{n(l-1)}$ avec l le nombre de niveaux.

Entrées	Nb tâches	Nb Comm.		Coût en calcul		Coût en Comm.		CCR
		moy	écart-type	moy	écart-type	moy	écart-type	
T_{small}	500	746	27	9,98	5.7	0.5	0.2	≈ 20
T_{moy}						5.1	2.5	≈ 2
T_{big}						10.2	5.1	≈ 1

TABLE 1 – Récapitulatif des caractéristiques des jeux de données de référence. Les coûts sont donnés en moyenne sur une tâche/arête.

4. Analyse de sensibilité

Dans cette section nous nous intéresserons à la sensibilité des algorithmes choisis selon deux caractéristiques : la distribution des coûts de calculs et la distribution des coûts de communications. Nous montrerons que ces modifications, qui peuvent sembler mineures, provoquent des changements importants en terme de performance. Nous avons omis les intervalles de confiance, trop petits pour être visualisés.

4.1. Performance des Groupes Témoins

Avant de réaliser l'analyse de sensibilité proprement dite, il nous faut observer la performance de nos algorithmes sur les groupes témoins.

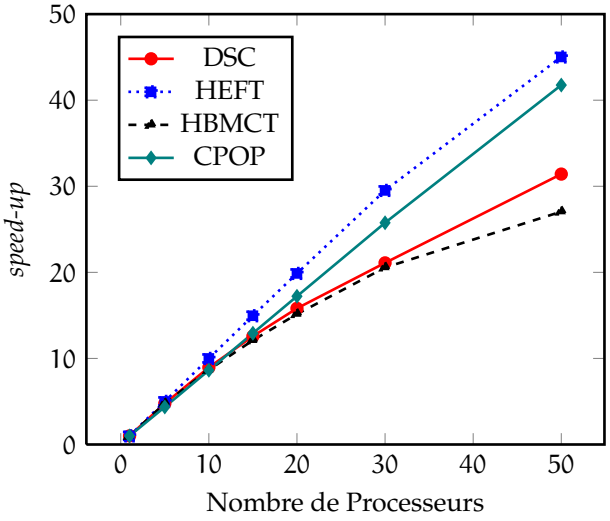

FIGURE 1 – Accélération des algorithmes en fonction du nombre de processeurs (sur T_{small}).

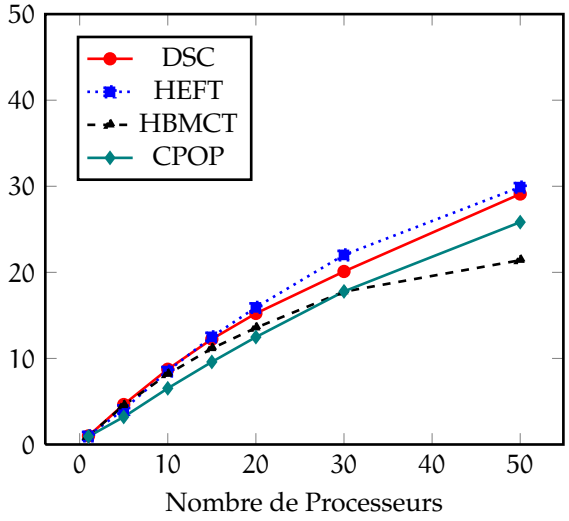

FIGURE 2 – Accélération des algorithmes en fonction du nombre de processeurs (sur T_{big}).

Les figures 1, 2 nous donnent respectivement les accélérations (*speed-up*) de nos algorithmes sur T_{small} et T_{big} avec un nombre de machines variant entre 1 et 50. Pour rappel, la loi de distribution utilisée pour les temps d'exécution et les communications est une loi uniforme. De plus, la plateforme utilisée est une clique (pas de contention). Le temps d'exécution moyen d'un DAG du groupe test est de 4995 sur une machine et d'environ 110 pour HEFT sur 50 machines.

Ces courbes mettent en valeur plusieurs caractéristiques des algorithmes choisis. Sur T_{small} avec un nombre de machines faibles (inférieur à 10), la plupart des algorithmes fournissent un *speed-up* quasi optimal. Lorsque le nombre de machines est proche au parallélisme de l'application (*speed-up* maximal), des différences de performance apparaissent. On peut aussi noter que HBMCT et DSC perdent en performance lorsque le nombre de machines est important.

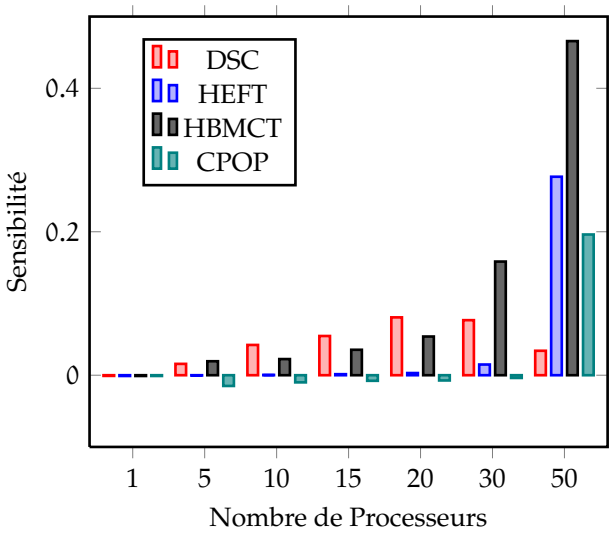

FIGURE 3 – Sensibilité au calcul en fonction du nombre de processeurs (sur T_{small}).

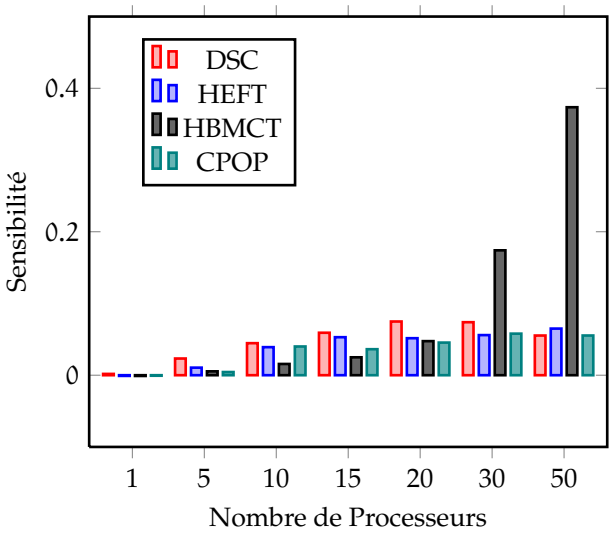

FIGURE 4 – Sensibilité au calcul en fonction du nombre de processeurs (sur T_{big}).

Lorsque les coûts en communications sont importants (comme sur T_{big} , avec CCR égal à 1), les différences de comportement entre algorithmes apparaissent plus nettement. DSC qui réalise une diminution importante du nombre de communications avec le *clustering* de tâches, est ainsi faiblement impacté. Au contraire, les algorithmes de liste voient leur efficacité décroître significativement avec le nombre de machines.

Les expériences sur T_{moy} montrent le même type de résultats.

Les comportements que nous venons d'observer nous servent de base pour analyser la sensibilité des algorithmes étudiés. Nous définissons la sensibilité d'un algorithme pour le reste de cet article comme la dégradation (ou amélioration) de sa performance entre son exécution sur le jeu d'entrées modifiées et celle sur le groupe témoin correspondant. Afin de simplifier l'interprétation de cette dégradation, nous la normalisons par la valeur de référence. Plus formellement, le calcul est le suivant :

$Sensibilité = \frac{C_{max} - C_{max\_ref}}{C_{max\_ref}}$ avec :

- C_{max} : le temps d'exécution sur jeu d'entrées modifiées.
- C_{max_ref} : le temps d'exécution obtenu sur le groupe test.

Ainsi, un algorithme avec un sensibilité de 0.1 sur une des modifications a vu son temps de complétion (observé sur le groupe modifié) augmenter de 10% par rapport au temps sur le groupe témoin.

4.2. Sensibilité à la Distribution des Calculs

Comme première étude de sensibilité, nous nous intéressons au comportement de nos algorithmes d'ordonnancement lorsque la distribution des coûts d'exécution des tâches change. Ce changement de la loi de distribution modifie la répartition entre petites tâches et grandes tâches (en temps d'exécution) dans les graphes en entrée. Nous avons choisi de passer ainsi d'une loi uniforme à une loi exponentielle (paramétrée pour conserver le même temps d'exécution moyen sur les tâches).

Aucun des algorithmes étudiés ne prends en compte cette distribution des coûts lors du calcul d'un d'ordonnancement. En conséquence, nous pourrions nous attendre à ce que tous réagissent, en moyenne, de la même façon face à ces modifications.

La figure 3 montre la sensibilité observée pour nos algorithmes sur une modification du groupe témoin avec peu de communications (T_{small}). Nous détaillons cette sensibilité en fonction du nombre de processeurs utilisés pour l'ordonnancement.

La sensibilité sur une machine (sans coûts de communications), nous confirme que les graphes donnés aux algorithmes possèdent bien une quantité de travail équivalente au groupe témoin (différence inférieure à 2%).

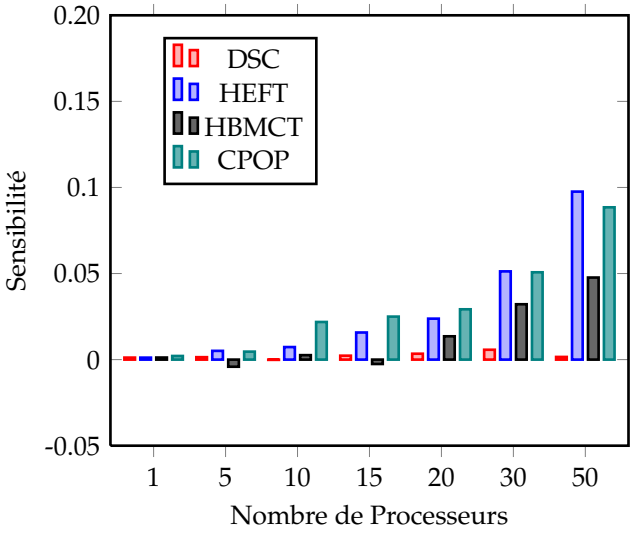

FIGURE 5 – Sensibilité aux communications en fonction du nombre de processeurs (sur T_{moy}).

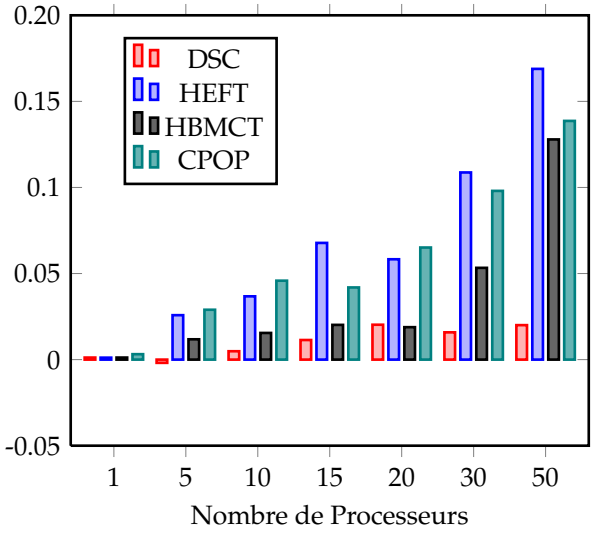

FIGURE 6 – Sensibilité aux communications en fonction du nombre de processeurs (sur T_{big}).

Face à notre jeu de données modifiées, trois phénomènes sont à observer. Premièrement, HBMCT subit une variation très importante de sa performance par rapport au groupe témoin (plus de 40 %). Cette sensibilité se manifeste surtout lorsque 50 machines sont utilisées et est négligeable en dessous du seuil de 30 machines. Ce seuil correspond à un nombre de machines équivalent au parallélisme des graphes en entrée (cf figure 1). Cela signifie que l'algorithme est instable quand il dispose de plus de machines que nécessaire pour calculer son ordonnancement. Cette sensibilité, que l'on retrouve dans une moindre mesure chez CPOP et HEFT entraîne un large rapprochement avec DSC qui est faiblement impacté par cette modification.

La figure 4 présente la même analyse de sensibilité avec une modification de T_{big}. De ce cas, les communications ont plus d'influence sur le temps de complétion que les temps d'exécution des tâches. Notons alors que la sensibilité des algorithmes diminue fortement, même si HBMCT reste plus sensible que les autres.

4.3. Sensibilité à la Distribution des Communications

Nous nous intéressons cette fois à la sensibilité des algorithmes choisis à la distribution des communications. Comme précédemment, nous avons modifié les groupes témoins pour passer d'une loi uniforme à une loi exponentielle.

La figure 5 montre l'évolution de la sensibilité en fonction du nombre de processeurs avec en entrée une modification de T_{moy}. Sur cette figure, la quantité de communications est peu élevée (CCR à 2). Dans ce cas, la sensibilité à la loi de distribution des communications est négligeable pour un nombre de machines inférieur à 30 sur l'ensemble des algorithmes. Sur un nombre de machines supérieur à 30, la sensibilité de DSC est largement plus faible voir même nulle, alors que les algorithmes de liste sont impactés à hauteur de 10%.

La figure 6 montre la même évolution lorsque les communications sont plus importantes (CCR à 1). Dans ce cas les algorithmes de *list-scheduling* sont significativement plus sensibles. Notons que DSC, un des algorithmes offrant de bonnes performances durant l'analyse des groupes témoins, est le moins sensible à cette modification des communications. Cela peut s'expliquer par un nombre de communications plus faible que les autres algorithmes. En effet, il regroupe en priorité les tâches qui communiquent le plus sur le même processeur et donc les communications les plus importantes seront rarement effectuées (impact faible).

Ces différences de sensibilité ne sont pas sans conséquences : dans certains cas, alors que l'analyse sur les groupes témoins nous donnait un algorithme comme étant meilleur qu'un autre, le jeu de données

modifié inverse cette relation.
Les figures 7, 8 montrent le *speed-up* observé sur les groupes modifiés tirés de T_{small} et T_{big} et mettent ces inversions en valeur, notamment entre DSC et HEFT sur des graphes avec beaucoup de communications.

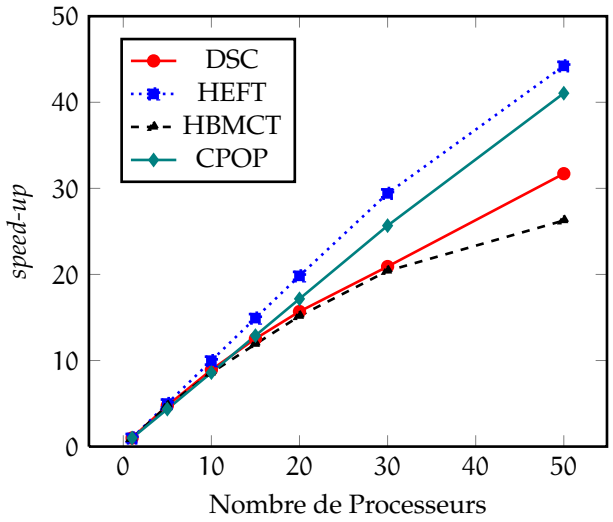

FIGURE 7 – Accélération des algorithmes en fonction du nombre de processeurs (sur T_{small} modifié).

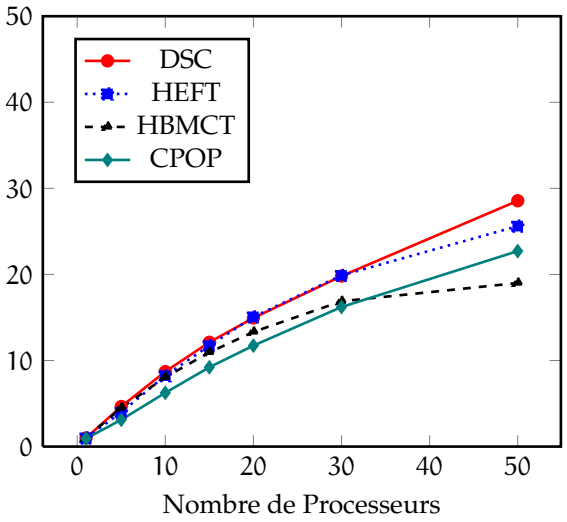

FIGURE 8 – Accélération des algorithmes en fonction du nombre de processeurs (sur T_{big} modifié).

Il est finalement intéressant d'observer que des algorithmes conçus pour résoudre des problèmes d'ordonnancement avec coûts de communications peuvent se montrer assez sensibles à la distribution de ces derniers. C'est notamment le cas lorsque le nombre de machines disponibles est plus important que la parallélisme des graphes en entrées. Enfin, la distribution des coûts de calculs semble elle aussi jouer un rôle, surtout lorsque les communications sont faibles.
Ces résultats ont aussi étés confirmés sur d'autres plateformes (prise en compte de la contention sur le réseau), d'autres types de graphes (plus d'arêtes) ou d'autres distributions (pareto).

5. Conclusion

Dans cet article, nous avons mis en valeur la sensibilité d'un ensemble d'algorithmes classiques de l'ordonnancement à des caractéristiques *fines* des graphes de tâches en entrée. En utilisant la simulation et des méthodes de génération de graphes contrôlées statistiquement nous avons montré que la distribution des temps de calculs ou des temps de communications jouait sur la qualité des ordonnancements obtenus.
Une des conséquences majeures de cette sensibilité est l'apparition d'inversions : alors que sur un jeu de données bien défini un algorithme semble plus performant qu'un autre, un autre jeu de données très proche du premier montre la comparaison inverse.
Ce résultat nous incite à poser un nouveau regard sur la comparaison d'algorithmes d'ordonnancement. En effet, la question de la sensibilité d'un algorithme par rapport aux jeux de données en entrée doit devenir centrale dans les comparaisons futures entre algorithmes d'ordonnancement.
Nous nous intéresserons lors de travaux futurs à une analyse plus détaillée de cette sensibilité, à l'aide de méthodes comme l'analyse factorielle et à son observation sur d'autres distributions ou lors de changements de plateforme d'exécution.

6. Remerciements

Les expériences présentées dans cet article ont étés réalisées à l'aide de la plateforme expérimentale Grid'5000, développée sous l'action du groupe INRIA ALADDIN avec le soutien du CNRS, de RENATER et plusieurs universités ainsi que d'autres fonds (voir `https://www.grid5000.fr`).

Bibliographie

1. Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid : a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008.
2. Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. Random graph generation for scheduling simulations. In *International ICST Conference on Simulation Tools and Techniques (SIMUTools)*, 2010.
3. David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. Logp : towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.
4. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software : Practice and Experience*, 30(11) :1203–1233, 2000.
5. Ronald L. Graham. Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math*, 17(2) :416–429, 1969.
6. Han J.A. Hoogeveen, jan K. Lenstra, and Bart Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, 16(3) :129 – 137, 1994.
7. Hans Meuer, Erich Strohmaier, Horst Simon, and Jack Dongarra. 35th release of the TOP500 list of fastest supercomputers, 2010.
8. R. Murphy, K. Wheeler, B. Barrett, and J. Ang. Introducing the Graph 500. 2010.
9. Sakellariou Rizos and Zhao Henan. A hybrid heuristic for dag scheduling on heterogeneous systems. In *International Parallel and Distributed Processing Symposium, 2004. Proceedings*, april 2004.
10. Yang Tao and Gerasoulis Apostolos. Pyrros : static task scheduling and code generation for message passing multiprocessors. In *Proceedings of the 6th international conference on Supercomputing*, ICS '92, pages 428–437, New York, USA, 1992. ACM.
11. Takao Tobita and Hironori Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5) :379–394, 2002.
12. Haluk Topcuouglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13 :260–274, March 2002.
13. Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33 :103–111, August 1990.
14. Tao Yang and Apostolos Gerasoulis. Dsc : Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5 :951–967, 1994.