Évaluation d'algorithmes d'ordonnancement par simulation réaliste

Adrien Faure, Millian Poquet, Olivier Richard

Compas'2018 : Parallélisme / Architecture / Système
ENSEEIHT - Toulouse, France, du 03 au 06 juillet 2018

Évaluation d'algorithmes d'ordonnancement par simulation réaliste

Adrien Faure[α,β], Millian Poquet[γ], Olivier Richard[β]

[α]Bull, Atos technologies
[β]Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
[γ]Univ Rennes, Inria, CNRS, IRISA, F-35000 Rennes, France

{adrien.faure, millian.poquet, olivier.richard}@imag.fr

Résumé
La diversité des plateformes de calcul à haute performance ne fait qu'augmenter. Le gestionnaire de ressources et de tâches (ou RJMS pour *Resources and Jobs Management Systems*) est responsable d'ordonnancer les tâches (applications souvent parallèles et distribuées) sur ce type de plateformes. Un ordonnancement mal maîtrisé peut dégrader significativement les performances des applications. Dans ce contexte, étudier et valider des RJMS ainsi que des algorithmes d'ordonnancement est un réel défi. Dans cet article nous présentons le logiciel Batsim, un simulateur d'infrastructure qui permet la simulation réaliste d'applications et l'étude de RJMS pour de nombreux problèmes. Nous validons notre approche en montrant que la prise en compte de la localité dans l'algorithme d'ordonnancement peut avoir un impact majeur sur les applications. Ce phénomène, ainsi que de nombreux autres, ne peuvent être observés qu'avec des modèles d'applications (et de plateformes) sensibles au contexte d'exécution.

Mots-clés : gestion de tâches et de ressources, ordonnancement, simulation

---

1. Introduction

Les gestionnaires de ressources et de tâches (ou RJMS pour *Resources and Jobs Management Systems*) jouent un rôle essentiel dans les plateformes modernes de calcul à haute performance, en permettant un partage équitable des ressources entre les utilisateurs tout en maximisant la productivité de la plateforme. Avec la volonté d'atteindre des puissances de calcul exaflopique, les systèmes de calcul à grande échelle deviennent de plus en plus complexe à étudier, à concevoir et à calibrer, à cause des forts coûts (financiers, temporels, énergétiques…) en présence.
Les évolutions des RJMS (ou de ses constituants, notamment les algorithmes d'ordonnancement) sont souvent étudiées puisqu'elles peuvent permettre des améliorations conséquentes sur divers objectifs. Étudier de telles évolutions directement sur des systèmes réels semble peu raisonnable tant les coûts et les risques impliqués sont forts. La simulation est donc en général préférée pour réaliser ces études, puisqu'elle permet un grand confort (déterminisme, reproductibilité, contrôle et observation parfaite du système, vitesse d'exécution). L'emploi de la simulation pose cependant un problème de réalisme dans ce domaine, puisque les modèles utilisés dans la plupart des simulateurs sont trop simples pour observer de nombreux phénomènes (contention réseau, hétérogénéité des ressources de calcul ou réseau, impact énergétique).

C'est dans cette optique que nous avons conçu le simulateur Batsim (pour ***Bat****ch scheduler* ***sim****ulator*), qui utilise le cadriciel de plateformes distribuées SimGrid [4]. Batsim propose différents niveaux de réalisme en fonction des besoins de l'utilisateur et suit une approche holistique — afin de permettre l'étude de nombreux problèmes, de nombreux objectifs, dans des scénarios aussi variés que possible. Batsim lit des formats d'entrée extensibles et produit des formats de sortie utilisables par les outils standards d'analyse, comprenant des informations sur les applications ou sur l'ordonnancement.
L'objectif de cet article est de présenter Batsim et les études qu'il aide à réaliser. La section 2 présente des travaux similaires autour de la simulation de RJMS. La section 3 présente le fonctionnement de Batsim. La section 4 montre que Batsim permet d'observer des phénomènes de contention réseau sur des plateformes de calcul. Enfin, nous concluons cet article en section 5.

2. État de l'art

L'utilisation d'un moteur de simulation de plateformes ou d'applications parallèles pour étudier des algorithmes d'ordonnancement existe dans la littérature [12, 2, 8]. Cependant, les modèles utilisés dans ces simulateurs ne permettent pas de modéliser les communications ni les calculs réalisés par les tâches, ce qui empêche d'observer l'impact de ces phénomènes.
Une autre approche intéressante est proposé dans [17]. Cette approche utilise le simulateur réseau à grain fin INSEE [18] de manière hors ligne afin de connaître précisément le temps d'exécution des jobs en fonction de la configuration dans laquelle ils sont lancés. Les temps obtenus sont alors utilisés dans une simulation en ligne avec des politiques de placement de tâches qui empêchent les interférences. Cette approche ne peut malheureusement pas être utilisée dans le cas général où les jobs peuvent interférer les uns avec les autres.
D'autres approches [20, 10] construisent des simulateurs autour de Slurm — un RJMS utilisé sur plusieurs machines du TOP500. L'utilisation de Slurm est pertinente car elle repose sur un système réel. Cependant ces approches reposent sur l'utilisation de machines virtuelles ou de l'émulation de plateformes pour simuler une plateforme réelle, ce qui passe peu à l'échelle si on souhaite étudier de grands systèmes. L'usage de Batsim permettrait, grâce à l'utilisation de SimGrid, d'explorer des simulations de plateformes à grande échelle [5]. De plus, l'implémentation d'algorithmes d'ordonnancement dans Slurm est difficile et ne permet pas la flexibilité proposée par Batsim.

3. Description de Batsim

3.1. Principe général

Batsim est un programme qui simule le comportement d'une plateforme de calcul parallèle sur laquelle des calculs sont exécutés. Batsim joue le rôle d'orchestrateur de la simulation entre l'ordonnanceur, la simulation des applications et la soumissions des applications dans le temps. Batsim connaît le temps courant de la simulation ainsi que l'ensemble des applications (terminées, en cours, et à venir). La simulation de l'exécution des applications sur la plateforme est effectuée via SimGrid [4]. SimGrid permet d'obtenir des temps de calcul en fonction de la puissance et de la charge des machines, ainsi que temps de communication en fonction de la latence, de la bande passante et de la charge des liens réseau. L'utilisation de SimGrid permet ainsi d'obtenir des résultats surs et d'étendre les possibilités de simulation, Batsim bénéficiant ainsi des améliorations faites dans SimGrid. La figure 1 illustre le fonctionnement de Batsim.
Toutes les décisions concernant l'ordonnancement d'applications ou le changement d'état des machines sont prises dans un processus séparé, appelé *processus de décision*. La communication

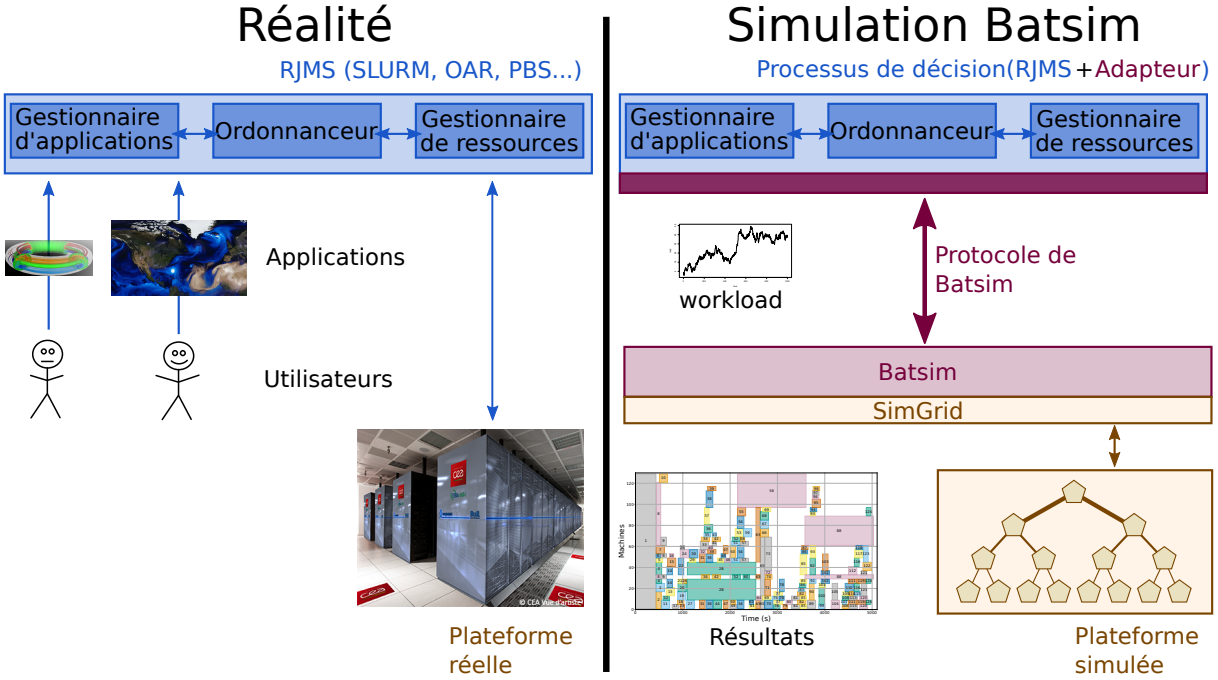

FIGURE 1 – Idée générale d'une simulation Batsim. Dans un vrai système, des utilisateurs veulent exécuter des applications sur une plateforme réelle. Avec Batsim, des applications définies dans des *workloads* sont soumises et leur exécution est simulée. Le même gestionnaire de ressources peut être utilisé dans les deux cas grâce à une couche d'adaptation.

entre les deux processus se fait via un protocole synchrone de type *question/réponse*. La *question* émise par Batsim sont des évènements de la simulation. La *réponse* attendue est une séquence d'actions à réaliser, par ex. le lancement d'une application sur certaines machines. En attendant que la réponse soit fournie par le *processus de décision*, Batsim met en pause la simulation. Une fois la réponse reçue, Batsim reprend la simulation en injectant lesdites actions en son sein. Cette forte séparation entre simulateur et algorithmes de décision permet d'utiliser lesdits algorithmes dans d'autres contextes, que ce soit avec d'autres simulateurs (par exemple dans WRENCH [21]) ou avec des RJMS de production (un prototype existe pour OAR [3]).
Batsim est un projet open source disponible sur différents dépôts [15, 16] dont le fonctionnement est testé par intégration continue. Plus de détails sur les motivations et les utilisations possibles de Batsim se trouvent dans la thèse [19].

3.2. Profils de simulation (modèles d'exécution d'applications)

Pour simuler l'exécution d'une application Batsim utilise un *profil* de simulation, ce qui permet de proposer différentes modélisations de la même application réelle. Le *profil* contient des informations qui décrivent comment l'application doit être simulée. Plus spécifiquement un *profil* instancie un modèle d'exécution de l'application (par ex. les communications qu'elle effectue, le montant de calculs). Trois principaux types de profil sont implémentés dans Batsim.
Le premier type de profil est le **délai**. Les application sont modélisées par un temps d'exécution fixe. Ce type de profil ne permet pas d'observer l'impact du contexte d'exécution (hétérogénéité de la plateforme, interactions entre les applications). Cependant il a l'avantage d'être rapide à simuler et très facile à instancier.

Le second type de profil est la **tâche parallèle**. Les applications sont modélisées par un vecteur de calcul et une matrice de communication. Le vecteur de calculs permet de spécifier la quantité de calcul (en nombre d'opérations flottantes) que chaque machine doit réaliser. La matrice de communications permet d'indiquer la quantité de données (en octets) que chaque machine doit envoyer à chacune des autres machines. Ce type de profil est un compromis entre vitesse de simulation et précision. Il permet de visualiser des effets de contention réseau.
Le dernier type de profil est **SMPI** [6]. Ce type de profil permet de simuler l'exécution d'une application MPI à partir d'une trace existante. Cette trace décrit la suite d'opérations que chaque machine doit exécuter. Les opérations peuvent être des calculs ou des appels de routines MPI.

4. Expérimentations sur des algorithmes

Afin d'illustrer l'utilisation de Batsim et ses bénéfices nous présentons dans cette section l'évaluation de deux algorithmes d'ordonnancement. L'expérience utilise des profils d'applications SMPI. Nous avons choisi d'utiliser SMPI car c'est le modèle le plus précis dont nous disposons dans Batsim pour observer les interférences réseau. La précision de SMPI a déjà été évaluée dans [6, 7]. Dans cette expérience nous comparons deux variantes d'un algorithme de remplissage (cf. section 4.3).

4.1. Modélisation de la plateforme

Afin de simuler les temps de calcul et de communication, Batsim a besoin d'information sur la plateforme de calcul. Nous avons ici choisi la grappe Graphene de l'infrastructure d'expérimentation Grid'5000 [1]. Le processus de modélisation et de calibration de la plateforme est décrit au lien suivant [11]. La plateforme est composée de quatre cabinets, chacun composé de 36 à 39 machines de calcul. Chacun de ces cabinets est relié aux autres via un lien de capacité 10 Gb/s. À l'intérieur d'un cabinet les nœuds sont reliés par un lien de 1 Gb/s. Le processus de calibration de la plateforme a permis de détecter les liens sujet à de la contention réseau. De la contention peut être observée à l'intérieur d'un cabinet, par ex. si tous les nœuds d'un même cabinet envoient d'importantes quantités de données au même nœud cible. Le second point de contention apparaît entre les différents cabinets, ce qui se produit lorsque de larges quantités de données sont échangées entre des nœuds répartis sur plusieurs cabinets.

4.2. Profils de simulation et charge de travail (*workload*)

Dans cette expérience nous injectons à Batsim une charge de travail (*workload*). La charge de travail est composée d'un ensemble d'applications. Nous avons généré une charge de travail de 512 applications de tailles (8 nœuds, 16 nœuds et 32 nœuds) en nous inspirant du chapitre 9.6 du livre [9]. Les dates de soumission suivent une distribution de Weibull. Les profils de simulation que nous avons utilisés sont des traces SMPI de l'application ft.C du NAS Parallel Benchmarks [14]. Nous avons exécuté ft.C avec plusieurs configurations (8 nœuds, 16 nœuds et 32 nœuds) afin d'en collecter des traces SMPI indépendantes du temps [6].

4.3. Algorithmes

Les deux algorithmes d'ordonnancement comparés dans cette expérience sont deux variantes d'un algorithme de remplissage. L'algorithme est en ligne, ce qui veut dire qu'il est appelé dès qu'une tâche est soumise ou termine. L'algorithme de remplissage sauvegarde les tâches soumises dans une file d'attente. À chaque appel, l'algorithme parcourt toute sa file d'attente (dans l'ordre de soumission) en démarrant les tâches qui peuvent être exécutées au temps courant.
Nous étudions deux variantes de l'algorithme de remplissage. Les deux variantes se distinguent

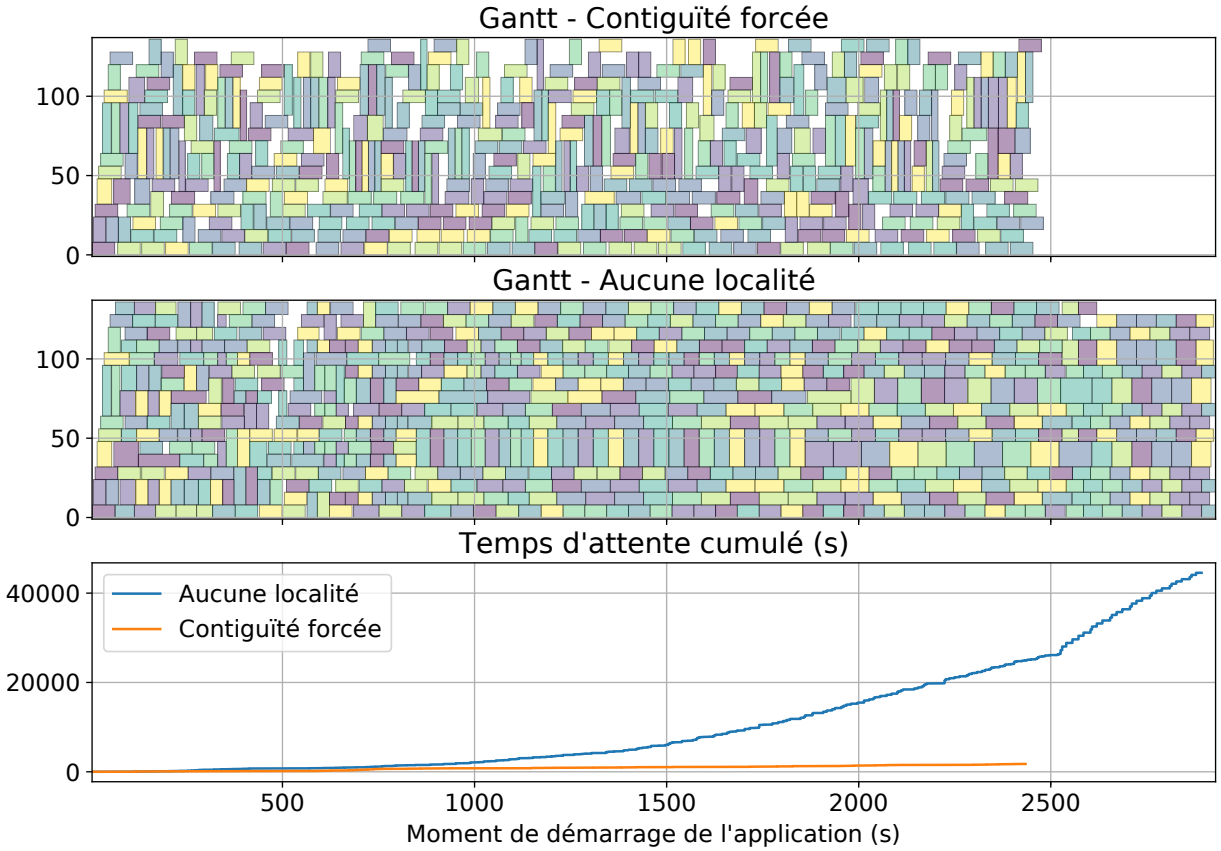

FIGURE 2 – Ordonnancements résultants des deux algorithmes étudiés.

par la contrainte de placement qu'elles imposent aux tâches [13]. En **contiguïté forcée**, les tâches doivent être placées sur des ressources contiguës. L'autre variante ne met **aucune** contrainte de placement sur les tâches.

4.4. Résultats et observations

Nous utilisons plusieurs métriques pour analyser ces résultats. Toute référence au temps dans cette section concerne le temps fictif passé en simulation (et non le temps d'exécution du simulateur). La première métrique est le *makespan*, qui correspond au temps de fin de la dernière tâche. Le *temps d'attente* d'une tâche est défini par l'intervalle de temps entre sa soumission et le début de son exécution. Le *temps d'exécution* d'une tâche correspond à l'intervalle de temps entre le début et la fin de son exécution. Le *facteur de localité* d'une tâche est défini par le rapport entre le nombre de cabinets utilisés pour son exécution et le nombre minimal de cabinets qu'elle aurait pu utiliser. Finalement le *stretch* d'une tâche est un rapport défini par la somme de son *temps d'attente* et de son *temps d'exécution*, le tout divisé par son *temps d'exécution*.

La figure 2 présente les deux diagrammes de Gantt issus des ordonnancements. Les deux diagrammes du haut présentent en ordonnées le nombre de ressources allouées et en abscisse le temps de la simulation. Le dernier diagramme représente la somme cumulée des *temps d'attentes* en seconde pour chacune des tâches. La figure 2 montre que Batsim permet d'observer une variations de performance entre les deux algorithme d'ordonnancement. Le *makespan* est la somme cumulée des *temps d'attentes* sont plus faibles lorsque la contiguïté est forcée.

Afin de mieux comprendre comment les tâches sont impactées par la politique de placement, la

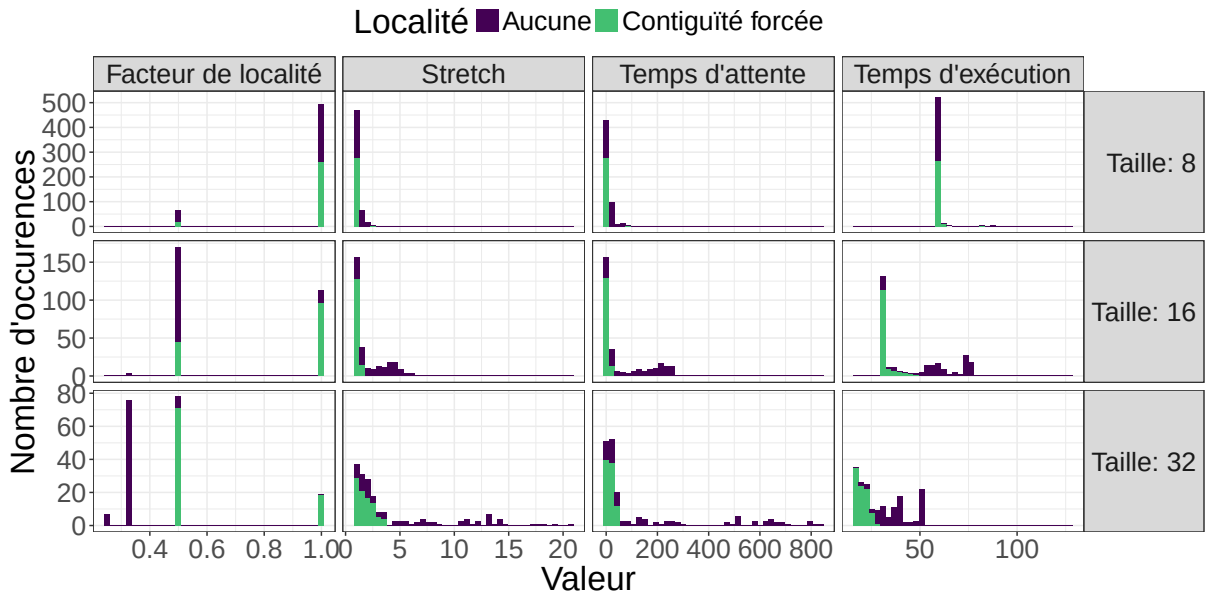

FIGURE 3 – Impact de la politique de localité sur différentes métriques, selon la taille des applications (en nombre de nœuds demandés).

figure 3 présente les histogrammes des différentes métriques pour chaque taille d'application. La colonne *temps d'exécution* montre que forcer les tâches à être placées de manière contiguë permet d'accélérer leur temps d'exécution. C'est le cas pour quelques tâches de taille 8 nœuds, mais on voit que la plupart des tâches de tailles 16 nœuds et 32 nœuds sont largement impactées par la contrainte de placement. Ceci s'explique par l'impact de la contrainte de placement sur la localité des tâches, comme le montre la colonne *facteur de localité*. Les tâches de taille 32 nœuds sont exécutées sur un ou deux cabinets lorsque la contiguïté est forcée, alors qu'elles sont la plupart du temps exécutées sur 3 cabinets ou plus sans contrainte de placement. Les tâches de taille 16 nœuds sont impactées de manière similaire, puisque forcer la contiguïté de leur placement augmente leur localité. Enfin, on voit que la localité des tâches de taille 8 nœuds est peu impactée par les contraintes de placement étudiées.

5. Conclusion

Cet article a présenté Batsim et une étude qui confirme que la prise en compte de la localité lors du placement d'applications est importante sur une plateforme de calcul distribuée. Cette étude est rendue possible grâce aux modèles d'exécution de Batsim qui permettent non seulement aux applications d'avoir un temps d'exécution différent selon leur placement, mais également de voir les interférences entre applications. Dans ce cas d'étude précis, les applications partagent des ressources réseau et on a pu y observer des effets de contention.
Batsim permet d'utiliser différents modèles d'applications et nous souhaiterions approfondir l'évaluation de chacun de ces modèles dans différents scénarios. Une étude approfondie permettra de connaître précisément les limitations des différents modèles d'exécution d'application selon le type d'étude visée — en particulier selon la précision avec laquelle chaque phénomène veut être observé.

Références

[1] Daniel BALOUEK et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In : *Cloud Computing and Services Science*. Sous la dir. d'Ivan I. IVANOV et al. T. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, p. 3–20. ISBN : 978-3-319-04518-4. DOI : 10.1007/978-3-319-04519-1_1.

[2] Yves CANIOU et J-S GAY. "Simbatch : An API for simulating and predicting the performance of parallel resources managed by batch systems". In : *Euro-Par 2008 Workshops-Parallel Processing*. Springer. 2009, p. 223–234.

[3] Nicolas CAPIT et al. "A batch scheduler with high level components". In : *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*. T. 2. IEEE. 2005, p. 776–783.

[4] Henri CASANOVA et al. "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms". In : *Journal of Parallel and Distributed Computing* 74.10 (juin 2014), p. 2899–2917. URL : http://hal.inria.fr/hal-01017319.

[5] Tom CORNEBIZE et al. "Emulating High Performance Linpack on a Commodity Server at the Scale of a Supercomputer". working paper or preprint. Déc. 2017. URL : https://hal.inria.fr/hal-01654804.

[6] Augustin DEGOMME et al. "Simulating MPI Applications : The SMPI Approach". In : *IEEE Transactions on Parallel and Distributed Systems* 28.8 (août 2017), p. 2387–2400. DOI : 10.1109/tpds.2017.2669305. URL : https://doi.org/10.1109%2Ftpds.2017.2669305.

[7] F. DESPREZ, G. S. MARKOMANOLIS et F. SUTER. "Improving the Accuracy and Efficiency of Time-Independent Trace Replay". In : *2012 SC Companion : High Performance Computing, Networking Storage and Analysis*. Nov. 2012, p. 446–455. DOI : 10.1109/SC.Companion.2012.64.

[8] Ahmed ELELIEMY, Ali MOHAMMED et Florina M. CIORBA. "Exploring the Relation between Two Levels of Scheduling Using a Novel Simulation Approach". In : *2017 16th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, juil. 2017. DOI : 10.1109/ispdc.2017.23. URL : https://doi.org/10.1109%2Fispdc.2017.23.

[9] Dror G. FEITELSON. *Workload Modeling for Computer Systems Performance Evaluation*. 1st. New York, NY, USA : Cambridge University Press, 2015. ISBN : 1107078237, 9781107078239.

[10] Rodrigo GONZALO P. et al. "ScSF : a scheduling simulation framework". In : *Proceedings of the 21th Workshop on Job Scheduling Strategies for Parallel Processing :* Lecture Notes in Computer Science. Work also supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) and we used resources at the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility, supported by the Officece of Science of the U.S. Department of Energy, both under Contract No. DE-AC02-05CH11231. 2017.

[11] *Graphene callibration*. http ://simgrid.gforge.inria.fr/contrib/smpi-saturation-doc.php.

[12] Dalibor KLUSÁČEK et Hana RUDOVÁ. "Alea 2 : job scheduling simulator". In : *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics et Telecommunications Engineering). 2010, p. 61.

[13] Giorgio LUCARELLI et al. "Contiguity and Locality in Backfilling Scheduling". In : *CCGrid 2015 - 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Shenzhen, China, mai 2015, p. 586–595. DOI : 10.1109/CCGrid.2015.143. URL : https://hal.archives-ouvertes.fr/hal-01230294.

[14] NASA. *NAS Parallel Benchmarks*. https://www.nas.nasa.gov/publications/npb.html. Fév. 2016. (Visité le 18/02/2016).

[15] OAR-TEAM. *Batsim Github Repository*. 2017. URL : https://github.com/oar-team/batsim (visité le 15/03/2017).

[16] OAR-TEAM. *Batsim Gitlab Repository*. 2017. URL : https://gitlab.inria.fr/batsim/batsim (visité le 15/03/2017).

[17] Jose A PASCUAL, Jose MIGUEL-ALONSO et Jose A LOZANO. "Locality-aware policies to improve job scheduling on 3D tori". In : *The Journal of Supercomputing* 71.3 (2015), p. 966–994.

[18] Fco Javier Ridruejo PEREZ et José MIGUEL-ALONSO. "INSEE : An interconnection network simulation and evaluation environment". In : *Euro-Par 2005 Parallel Processing*. Springer, 2005, p. 1014–1023.

[19] Millian POQUET. *Simulation Approach for Resource Management*. https://tel.archives-ouvertes.fr/tel-01757245v1. 2017.

[20] Nikolay A. SIMAKOV et al. "A Slurm Simulator : Implementation and Parametric Analysis". In : *Lecture Notes in Computer Science*. Springer International Publishing, déc. 2017, p. 197–217. DOI : 10.1007/978-3-319-72971-8_10. URL : https://doi.org/10.1007%2F978-3-319-72971-8_10.

[21] *Site web du projet WRENCH*. http://wrench-project.org.