RenPar'17 / SympA'2006 / CFSE'5 / JC'2006
Canet en Roussillon, 4 au 6 octobre 2006

Simbatch : une API pour la simulation et la prédiction de performances de systèmes batch

Jean-Sébastien Gay, Yves Caniou

LIP, ENS-Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07
Jean-Sebastien.Gay@ens-lyon.fr

LIP, ENS-Lyon, Université Claude Bernard Lyon 1
Yves.Caniou@ens-lyon.fr

Résumé
Les études d'algorithmes d'ordonnancement de tâches parallèles dans le contexte des grilles de calcul ignorent souvent les systèmes de réservation locaux qui gèrent les ressources parallèles, ou supposent qu'ils instancient *First Come First Served*. Nous décrivons donc dans cet article une API intégrée au simulateur de grille Simgrid. Elle offre les structures et fonctionnalités pour simuler de façon très réaliste les grappes de PCs et les systèmes de réservation batch pour les gérer. Les expériences montrent des erreurs de simulation inférieures à 1% par rapport aux résultats *réels* obtenus avec le système de réservation OAR.

Mots-clés : Ordonnancement, Simulation de grilles de calcul, Systèmes batch, Prédiction de performances

---

1. Introduction

La fédération des ressources de calcul requière un travail sur la mise à disposition de leurs puissances, qui peut se faire à l'aide d'intergiciels de grille comme DIET[4] ou NetSolve[5]. Leur objectif est de proposer aux différents utilisateurs de la grille la résolution de problèmes de calcul, tout en cachant la complexité de la plate-forme et de ses accès.
Actuellement, les grilles de calcul sont construites sur un modèle assez proche d'une hiérarchie de grappes de clusters. Ainsi en est-il de la plate-forme de production du projet EGEE[1] (*Enabling Grids for E-science in Europe*) qui regroupe plus d'une centaine de centres répartis dans 31 pays ou de la grille de calcul française pour la recherche Grid'5000[3], dont le but à terme est de disposer de 5000 nœuds répartis sur la France (actuellement, 9 centres y participent).
Généralement, l'utilisation d'une ressource de calcul parallèle se fait via un système de réservation batch : les tâches parallèles qu'un utilisateur souhaite exécuter lui sont soumises à l'aide de *scripts* décrivant notamment la durée estimée et le nombre de nœuds souhaités. L'algorithme d'ordonnancement du système batch s'applique ensuite pour déterminer la date d'exécution et l'identité des nœuds affectés pour chaque tâche.
L'ordonnancement apparaît donc au moins à deux niveaux : celui du système batch et celui de l'intergiciel de grille. Afin d'en exploiter les ressources le mieux possible, l'intergiciel doit affecter les tâches de calcul en accord avec les politiques d'ordonnancement locales. La conception de tels algorithmes est non triviale et leur validation pose problème. La réalisation d'expériences *dimensionnantes*, lorsqu'elles sont possibles, monopolise les ressources. Il faut définir des bases communes pour les simuler afin de mesurer les performances des algorithmes proposés avant tests réels. Nous proposons donc ici un moyen facile et réutilisable intégré comme le premier module au simulateur de grille Simgrid, dans le but de réaliser de telles études.

Les contributions de ce travail sont principalement la conception d'une API intégrée à Simgrid afin de simuler facilement des grappes de PCs et les comportements des systèmes batch dans une grille de calcul.

[1] http://public.eu-egee.org/

Ainsi, les études d'algorithmes d'ordonnancement pour un intergiciel de grille, ou même de nouveaux algorithmes d'ordonnancement batch, sont grandement simplifiées. De plus, les modélisations de PBS et d'OAR[2] réalistes sont proposées. L'excellente qualité des résultats des simulations effectuées lors de la validation de l'API permet d'envisager son utilisation pour effectuer de la prédiction de performances, nécessaire à l'intergiciel pour ordonnancer ses tâches de calcul.

Cet article est organisé de la façon suivante : nous présentons d'abord le contexte de travail dans la section 2 ; nous décrivons l'API proposée et les modèles utilisés à la section 3. Les expériences réelles et de simulation sont exposées à la section 4 et les résultats sont analysés et commentés dans la section 5. Nous concluons à la section 6 en présentant nos perspectives.

2. Les simulateurs de grille

Il existe de nombreux simulateurs de grille parmi lesquels nous pouvons citer Bricks[10], pour la simulation de systèmes du type client-serveur, OptorSim[1], conçu pour l'étude d'algorithmes d'ordonnancement traitant spécifiquement de la réplication ou de la migration de données, ou encore Simgrid[7]. Ce dernier offre une API permettant de modéliser simplement les ressources d'une grille de calcul pour tester des algorithmes d'ordonnancement centralisé ou distribué.
Les travaux les plus proches de notre travail sont [6] et [9]. Pour effectuer leurs études, les auteurs modélisent des ordonnanceurs locaux. Cependant, les systèmes utilisés ne sont pas génériques : ils ne proposent pas une API de fonctions réutilisables ; par soucis de simplification, l'ordonnancement utilisé est *First Come-First Served* ; enfin, *les tâches parallèles* ne sont pas supportées.
Les études menées sur l'ordonnancement au niveau de la grille de calcul ne sont donc pas nécessairement réalistes à cause de la politique d'ordonnancement locale des systèmes parallèles simulés. En effet, la figure 1 montre la différence de *flow* (temps passé dans le système, c'est-à-dire le temps d'attente ajouté aux temps d'exécution et de communication) de 100 tâches d'une même expérience soumise à une ressource parallèle composée de 7 machines, contrôlée respectivement par *FCFS* (implanté dans PBS[3]) et *Conservative Backfilling*, implanté dans OAR. On peut aussi noter la différence de temps de terminaison des expériences, respectivement 83297 secondes pour FCFS contre 74270 pour Conservative Backfilling.
Nous proposons donc Simbatch[4], une API pour simuler facilement de tels systèmes. Nous avons choisi de l'intégrer à Simgrid car le projet est très réactif et son utilisation croissante dans la communauté.

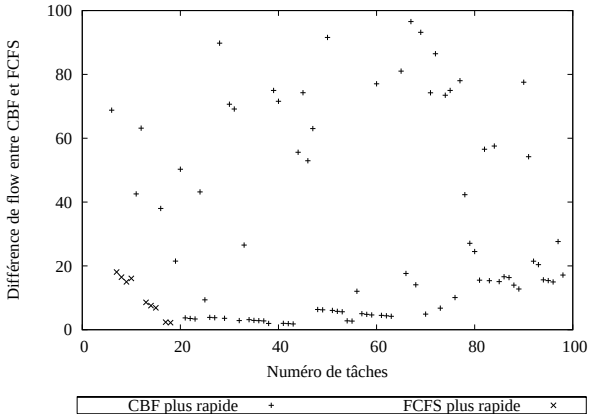

FIG. 1 – Différence de flow observé pour 100 tâches entre FCFS et Conservative backfilling

[2] http://oar.imag.fr
[3] http://www.openpbs.org
[4] http://simgrid.gforge.inria.fr/doc/contrib.html

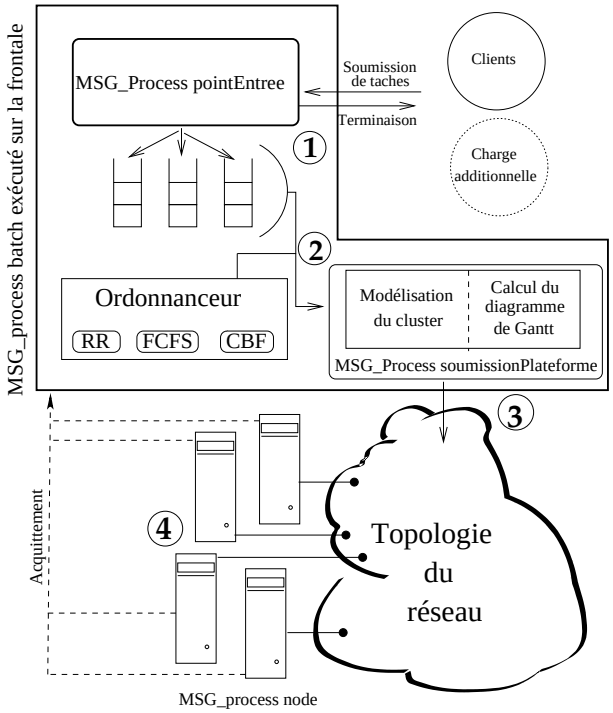

FIG. 2 – Fonctionnement de Simbatch

Tâches	1	2	3	4	5
Nombre de processeurs	1	5	2	1	3
Date de soumission	0	600	1800	3600	4200
Durée de la tâche	10800	3300	5400	4000	2700
Durée de réservation	12000	4000	7000	5000	3500

FIG. 3 – Données de l'expérience 1

3. Une API pour simuler des systèmes de réservation

3.1. Présentation de Simbatch

Simbatch est une API représentant environ 2000 lignes de code C. Elle a pour but de faciliter la conception et l'évaluation d'algorithmes d'ordonnancement à l'échelle de la grille ou d'un cluster. Elle utilise les structures et les fonctionnalités de la bibliothèque Simgrid pour modéliser des grappes de PCs et pour décrire des systèmes de réservation batch nécessaires à leur gestion.

Simbatch fournit une bibliothèque contenant déjà trois algorithmes d'ordonnancement[8] qui sont : *Round Robin* (RR), *First Come First Served* (FCFS) et *Conservative BackFilling* (CBF). L'API permet à l'utilisateur d'en ajouter d'autres aisément. Afin de pouvoir observer le comportement des algorithmes, une sortie compatible avec le logiciel Pajé[5] pour visualiser les diagrammes de Gantt peut être produite.

3.2. Modélisation

Une grappe est composée d'une frontale et de ressources de calcul interconnectées selon une topologie déterminée. Le système batch est exécuté sur la frontale. Il reçoit les requêtes des utilisateurs et doit les ordonnancer sur les ressources de calcul. Cette opération consiste à déterminer l'identité des noeuds affectés à chaque tâche de calcul et la date d'exécution. Les tâches de calcul sont parallèles, prennent des données en entrée et en produisent en sortie.

Dans Simbatch, une tâche parallèle soumise par un client est modélisée par l'adjonction d'informations à une tâche de calcul Simgrid, comme le nombre de processeurs, la durée d'exécution et de réservation. Les autres modèles sont directement issus de Simgrid.

Comme le montre la figure 2, le traitement des tâches s'effectue de la façon suivante :

1. Le processus `pointEntree` accepte les soumissions de tâches parallèles des différents clients et les place dans la file d'attente appropriée.
2. Grâce à l'unité de modélisation, l'ordonnanceur attribue une date d'exécution aux tâches et réserve

[5] `http://www-id.imag.fr/Logiciels/paje/`

les ressources nécessaires. Une vue d'ensemble de la grappe est obtenue grâce au diagramme de Gantt ainsi calculé.

3. Le module de soumission gère l'envoi de chaque tâche à la date déterminée sur les ressources réservées. Il contrôle également le bon respect des réservations : une tâche est tuée si la date d'échéance est dépassée.
4. Simbatch repose sur Simgrid pour la simulation des communications et des exécutions des tâches de calcul. Quand une exécution termine, un acquittement est envoyé au processus `batch` pour qu'il mette à jour la modélisation de la grappe.

3.3. Instancier des systèmes de réservation avec Simbatch

Simgrid utilise un *fichier de description de la plate-forme* simulée. L'ensemble des ressources (calcul et réseau) ainsi que la connectivité, y sont définis. Ainsi, il contient les ressources décrivant les différentes grappes participant à la simulation.
Simbatch requiert un *fichier de déploiement*. L'utilisateur y fournit le nom des frontales qui accepteront les soumissions de tâches parallèles dans une ligne du type :

```
<process host="Frontale" function="batch"/>
```

Cette ligne indique que l'hôte `Frontale` exécutera le processus `batch` offert par l'API Simbatch. De même, les ressources de calcul de chaque grappe y sont déclarées avec le processus `node` offert par Simbatch qu'elles doivent exécuter. Pour chaque frontale présente dans la plate-forme, il faut aussi une description dans un *fichier de configuration* au format XML. L'utilisateur y spécifie le nombre de files d'attente utilisées ainsi que l'algorithme d'ordonnancement utilisé (voir p.8).
Il est également possible de simuler une *charge interne* pour *chaque* système de réservation. Elle est stockée dans un fichier dont le nom est donné dans le fichier de configuration. Il contient les caractéristiques des tâches, dont la date à laquelle chacune sera soumise.
À titre d'exemple, l'annexe contient les fichiers à partir desquels les expériences présentées plus loin ont été réalisées. Le code principal montre un client soumettant une tâche à un système de réservation ayant une charge interne, doté de l'ordonnancement CBF, contrôlant 3 files de priorité et 5 nœuds directement connectés à la frontale.

4. Expérimentations

4.1. Génération des tâches

Afin de valider les résultats obtenus par Simbatch, nous avons conçu à l'aide de la bibliothèque GSL[6] un générateur de charge. Celui-ci utilise une distribution de Poisson pour générer les temps d'inter-arrivées. Les caractéristiques des tâches sont déterminées par des lois uniformes. Ainsi, le nombre de processeurs utilisés est donné par $U(1;5)$, la durée de calcul par $U(600;1800)$ et la durée de réservation est obtenue en pondérant la durée de calcul par un nombre généré par $U(1.1;3)$.

4.2. Plate-forme d'expérimentation réelle

Le système de réservation batch OAR[2] est développé à Grenoble. C'est le système de réservation déployé sur chaque site de la plate-forme Grid'5000. L'algorithme d'ordonnancement implanté est CBF.
La version 1.6 d'OAR a été installée sur un cluster composé d'un serveur frontale et de 5 serveurs SuperMicro 6013PI dotés d'un processeur Xeon à 2.4 GHz, tous réliés à un switch à 100 Mbits/sec.

4.3. Protocoles d'expérimentation

Deux protocoles d'expérimentation ont été utilisés pour valider notre travail.
Pour le premier protocole, nous avons généré plusieurs clusters en faisant varier le nombre de nœuds et les topologies réseau utilisées : bus, étoile et mixte. Pour chaque cluster, un système batch a été généré à partir de l'un des trois algorithmes d'ordonnancement fournis et un nombre de files d'attente tiré aléatoirement.
Pour le deuxième protocole, nous avons modélisé la plate-forme réelle par des ressources de calcul interconnectées en étoile. Grâce au générateur de charge, nous avons ensuite soumis les mêmes charges à la

[6] `http://www.gnu.org/software/gsl/`

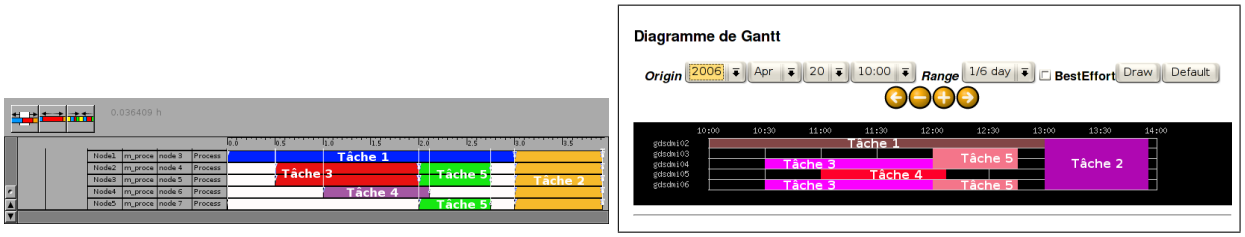

FIG. 4 – Diagramme de Gantt pour l'expérience 1 : à gauche avec Simbatch, à droite avec OAR

plate-forme réelle et à la plate-forme simulée. Pour cela, nous avons créé à l'aide de la biliothèque MPI, une tâche de calcul dont la durée est passée en paramètre. Cette tâche est exécutée entre deux appels de mesure de temps dans dans un script OAR, effectués pour déterminer les dates d'exécution et de terminaison. La précision de la mesure est de l'ordre de 1 seconde, donc négligeable devant la durée des tâches.

5. Résultats et discussions

5.1. Validation des algorithmes d'ordonnancement

Pour chaque instance (cluster,batch) du premier protocole, de nombreuses expériences ont été réalisées à l'aide du générateur de charge décrit avant. Ainsi, les trois algorithmes RR, FCFS et CBF fournis par la bibliothèque Simbatch ont fait l'objet de nombreux tests de simulation pour valider l'exactitude de leurs résultats.

Nous présentons dans la figure 4 le résultat obtenu pour l'une des expériences du deuxième protocole d'expérimentation, décrite dans le tableau de la figure 3. Elle montre, à gauche, le diagramme de Gantt obtenu avec Pajé et, à droite, celui obtenu avec l'outil DrawGantt d'OAR.

L'ordre d'exécution des tâches est rigoureusement le même : les tâches 3, 4 et 5 bénéficient bien du *backfilling* et commencent donc leur exécution avant la tâche 2 qui demande tous les nœuds du cluster. On peut cependant remarquer que pour la tâche 3, Simbatch n'alloue pas nécessairement aux tâches les mêmes nœuds qu'OAR.

5.2. Précision des simulations Simbatch

Nous présentons ici une expérience représentative de celles du deuxième protocole. Elle consiste en la soumission de 35 tâches de calcul : l'expérience soumise à OAR a duré 29242 secondes (environ 8 heures) et 29165 secondes en simulation. La différence de makespan (temps total d'exécution) est de 77 secondes, ce qui représente un taux d'erreur de 0,2% et correspond environ à l'imprécision de la prise de mesure multipliée par le nombre de tâches soumises.

La figure 5 montre le taux d'erreurs observé en fonction des dates *d'exécution* des tâches. On remarque que le taux d'erreurs est constant et inférieur à 1%, ce qui est négligeable au vu de la précision de la mesure.

La figure comporte également une flèche située au temps 10287, qui représente la date de la dernière *soumission*. Dans un cadre dynamique, si on fournit à Simbatch les caractéristiques des tâches soumises à un système de réservation, il est donc capable d'effectuer une prédiction fiable sur l'ensemble des tâches soumises plusieurs heures à l'avance *sans mécanisme de synchronisation*. En plus d'obtenir une simulation très réaliste, ceci peut s'avérer utile dans le module de prédiction de performances d'un intergiciel de grille par exemple.

Simbatch permet donc de modéliser simplement des ressources parallèles gérées par un système batch. La qualité de ses estimations montre que son utilisation pour l'étude d'algorithmes d'ordonnancement pour la grille est pertinente.

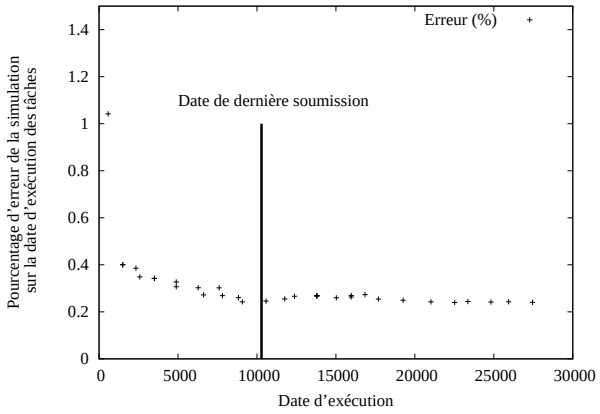

FIG. 5 – Taux d'erreurs d'une simulation Simbatch par rapport à la réalité avec OAR

6. Conclusions et travaux futurs

Les simulateurs de grille actuels ne permettent pas de tester facilement de nouveaux algorithmes d'ordonnancement pour la grille : il manque dans leurs API les fonctionnalités permettant la mise en œuvre rapide de systèmes batch généralement présents sur les ressources parallèles. C'est pourquoi nous proposons dans cet article l'API Simbatch, le premier module intégré à Simgrid.

Simbatch est un outil qui peut être utilisé pour la conception d'algorithmes d'ordonnancement pour les systèmes batch et pour l'étude des performances d'algorithmes d'ordonnancement dynamiques à l'échelle de la grille, comme celui d'un intergiciel devant prendre en compte les politiques d'ordonnancement locales à chaque cluster.

Nous avons détaillé ses fonctionnalités en précisant les modèles utilisés. Puis nous avons montré la facilité de prise en main pour tout utilisateur de Simgrid, à l'aide des exemples utilisés pour valider notre travail. Les ordonnancements principaux (*Round Robin*, *First Come First Served* et *Conservative BackFilling*) ont été implantés et validés par de nombreuses expériences de simulation. De plus, nous avons effectué des expériences réelles avec OAR afin de montrer d'une part que la mise en place d'un tel système batch dans Simgrid est maintenant rapide et d'autre part, que la précision de la simulation obtenue est excellente.

Ce travail nous ouvre de nombreuses perspectives : la conception de Simbatch vise à nous fournir un moyen pratique et facile pour la conception et l'étude d'algorithmes d'ordonnancement dynamiques qui seront intégrés dans l'intergiciel de grille hiérarchique DIET. Les ressources de Grid'5000 pourront être facilement modélisées et des expériences de simulation conduites, afin de sélectionner les expériences dimensionnantes réelles à tester.

De plus, la soumission de tâches parallèles par un intergiciel est non triviale en particulier en raison du manque de fonction d'estimation dans les systèmes batch. Grâce à Simbatch, il est possible d'utiliser une modélisation suffisante de ces systèmes afin de pouvoir déterminer le nombre de processeurs à utiliser, dans quelle queue soumettre, quelle durée de réservation demander pour obtenir la date d'exécution la plus tôt possible par exemple. L'intergiciel peut ensuite déterminer quelle ressource parallèle utiliser et avec quels paramètres soumettre la tâche de calcul. Nous réfléchissons donc à l'intégration de Simbatch dans le module de prédiction de performances des serveurs DIET déployés sur les frontaux des ressources parallèles.

Bibliographie

1. W. Bell, D. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Optorsim - a grid simulator for studying dynamic data replication strategies. *Journal of High Performance Computing Applications*, 17, 2003. `https://edms.cern.ch/file/392802/1/OptorSimIJHPCA2003.ps`.
2. N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounier, P. Neyron, and O. Richard.

A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid05)*, 2005. http://oar.imag.fr/papers/oar_ccgrid05.pdf.

3. F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet. Grid'5000 : A large scale, reconfigurable, controlable and monitorable grid platform. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, Grid'2005*, Seattle, Washington, USA, November 2005.
4. E. Caron, F. Desprez, E. Fleury, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. *Calcul réparti à grande échelle*, chapter Une approche hiérarchique des serveurs de calculs. Hermès Science Paris, 2002. ISBN 2-7462-0472-X.
5. H. Casanova and J. Dongarra. Netsolve : A network server for solving computational science problems. In *Proceedings of Super-Computing -Pittsburg*, 1996.
6. V. Garonne. *DIRAC - Distributed Infrastructure with Remote Agent Control.* PhD thesis, Université de Méditéranée, Décembre 2005.
7. A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications : the simgrid simulation framework. In IEEE Computer Society, editor, *3rd International Symposium on Cluster Computing and the Grid*, page 138. IEEE Computer Society, May 2003.
8. A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. In IEEE, editor, *IEEE Trans. Parallel and Distributed Systeme*, volume 12, pages 529–543. IEEE, June 2001. http://www.cs.huji.ac.il/~feit/papers/SP2backfil01TPDS.ps.gz.
9. K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, 2002. http://www.cs.uchicago.edu/%7Ekrangana/papers/decoupling_comp_data.ps.
10. A. Takefusa. Bricks : A performance evaluation system for scheduling algorithms on the grids. In *JSPS Workshop on Applied Information Technology for Science (JWAITS 2001)*, 2001.

Annexe

```
<?xml version='1.0'?>
<!DOCTYPE platform_description SYSTEM "surfxml.dtd">
<platform_description>
  <process host="Client" function"client">
     <argument value="0" />
     <argument value="0" />
     <argument value="0" />
     <argument value="Frontale" /> <!-- Connection -->
  </process>
  <!-- The Scheduler process (with some arguments) -->
  <process host="Frontale" function="SB_batch">
     <argument value="0" />      <!-- Number of tasks -->
     <argument value="0" />      <!-- Size of tasks -->
     <argument value="0" />      <!-- Size of I/O -->
     <argument value="Node1" /> <!-- Connections -->
     <argument value="Node2" />
     <argument value="Node3" />
     <argument value="Node4" />
     <argument value="Node5" />
  </process>
  <process host="Node1" function="SB_node"/>
  <process host="Node2" function="SB_node"/>
  <process host="Node3" function="SB_node"/>
  <process host="Node4" function="SB_node"/>
  <process host="Node5" function="SB_node"/>
</platform_description>
```

Fichier de déploiement

```
<?xml version='1.0'?>
<!DOCTYPE platform_description SYSTEM "surfxml.dtd">
<platform_description>
  <cpu name="Client" power="97.34000000000000"/>
  <!-- One scheduler for one cluster of five nodes -->
  <!-- Power of the batch is not important         -->
  <cpu name="Frontale" power="98.094999999999999"/>
  <cpu name="Node1" power="76.296000000000006"/>
  <cpu name="Node2" power="76.296000000000006"/>
  <cpu name="Node3" power="76.296000000000006"/>
  <cpu name="Node4" power="76.296000000000006"/>
  <cpu name="Node5" power="76.296000000000006"/>
  <!-- No discrimination for the moment -->
  <network_link name="0" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="1" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="2" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="3" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="4" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="5" bandwidth="41.279125" latency="5.9904e-05"/>
  <!-- Simple topologie -->
  <route src="Client" dst="Frontale"><route_element name="0"/></route>
  <route src="Frontale" dst="Node1"><route_element name="1"/></route>
  <route src="Frontale" dst="Node2"><route_element name="2"/></route>
  <route src="Frontale" dst="Node3"><route_element name="3"/></route>
  <route src="Frontale" dst="Node4"><route_element name="4"/></route>
  <route src="Frontale" dst="Node5"><route_element name="5"/></route>
  <!-- Bi-directionnal -->
  <route src="Node1" dst="Frontale"><route_element name="1"/></route>
  <route src="Node2" dst="Frontale"><route_element name="2"/></route>
  <route src="Node3" dst="Frontale"><route_element name="3"/></route>
  <route src="Node4" dst="Frontale"><route_element name="4"/></route>
  <route src="Node5" dst="Frontale"><route_element name="5"/></route>
</platform_description>
```

Fichier de description de la plate-forme

```
<?xml version="1.0" ?>

<config>
        <!-- Global settings for the simulation -->
        <global>
                <file type="platform">platform.xml</file>
                <file type="deployment">deployment.xmk</file>
                <!-- Paje output : suffix has to be .trace -->
                <file type="trace">simbatch.trace</file>
        </global>

        <!-- Each batch deployed should have its own config -->
        <batch host="Frontale">
                <plugin>librrobin.so</plugin>
                <!-- Internal Load -->
                <wld>./workload/seed/1.wld</wld>

                <priority_queue>
                        <number>3</number>
                </priority_queue>
        </batch>

        <!-- Other batchs -->
</config>
```

Configuration du système batch simulé

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <msg/msg.h>
#include <simbatch.h>
#define NB_CHANNEL 10000

/* How to create and send a task */
int client(int argc, char * argv)
{
    job_t job=calloc(job_t, sizeof(job));
    m_task_t task=NULL;

    strcpy(job->name, "tache");
    job->nb_procs = 3; job->priority = 1;
    job->wall_time = 600; job->requested_time = 1800;
    job->input_size = 100; job->output_size = 600;
    task = MSG_task_create(job->name, 0, 0, job);
    MSG_task_put(task,MSG_get_host_by_name("Frontale")
                 ,CLIENT_PORT);
}

int main(int argc, char ** argv)
{
  SB_global_init(&argc, argv);
  MSG_global_init(&argc, argv);

  /* Open the channels */
  MSG_set_channel_number(NB_CHANNEL);
  MSG_paje_output("simbatch.trace");

  /* The client who submits requests (write your own)
   * Params have to be called with the same name */
  MSG_function_register("client", client);

  /* Register simbatch functions */
  MSG_function_register("SB_batch", SB_batch);
  MSG_function_register("SB_node", SB_node);

  MSG_create_environment("platform.xml");
  MSG_launch_application("deployment.xml");

  MSG_main();

  /* Clean everything up */
  SB_clean();
  MSG_clean();

  return EXIT_SUCCESS;
}
```

Code principal Simgrid