Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems

Y. Caniou[1,⋆,⋆⋆] and J.-S. Gay[2,⋆,⋆⋆,⋆⋆⋆]

[1] LIP-ÉNS de Lyon, Université Claude Bernard de Lyon
yves.caniou@ens-lyon.fr
[2] LIP-ÉNS de Lyon
Jean-Sebastien.Gay@ens-lyon.fr

Abstract. In this paper, we describe Simbatch, an API which offers core functionalities to realistically simulate parallel resources and batch reservation systems. The objective is twofold: proposing at the same time a tool to efficiently predict parallel resources usage based on their simulations, and to realistically study Grid scheduling heuristics that may be embedded in a Grid middleware or in a tool that deploys it. Indeed, such predictions can be used in a Grid middleware both for scheduling purposes, and to dynamically tune moldable applications in function of the load of the chosen parallel resource in place of the Grid user. Simbatch simulation experiments show an average error rate under 2% compared to *real life* experiments conducted with the OAR batch manager.

Keywords: Performance prediction, Batch systems simulation, Grid simulation, Scheduling.

1 Introduction

Nowadays Grids are built on a clusters hierarchy model, as used for example by the two projects EGEE [1] and GRID'5000 [9]. The production platform in the EGEE project (*Enabling Grids for E-science in Europe*) aggregates more than 100 sites spread over 31 countries. GRID'5000 is the French Grid for the research, which aims to own 5000 nodes spread over France (9 sites are actually participating).

Parallel computing resources are generally managed via a batch reservation system also called *batch scheduler*. Users wishing to submit parallel tasks to the resource have to write *scripts* which notably describes the number of required nodes and the walltime of the reservation. They are generally answered the starting time of their jobs.

The accessibility to the aggregated power of a federation of computing resources requires mechanisms to monitor, handle/submit jobs, etc. This can be done with the help of Grid middleware such as DIET [10] or NetSolve [11]. They aim to offer to Grid users

⋆ This work is supported by the LEGO project ANR-05-CIGC-11.
⋆⋆ This work is supported by the REDIMPS project JST-CNRS.
⋆⋆⋆ This work is supported by the cluster Rhône Alpes
[1] http://public.eu-egee.org/

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 223–234, 2009.

the capacity to efficiently solve problems, while hiding the complexity of the platform. In order to efficiently exploit the resource, Grid middleware should map the computing tasks according to local scheduler policy and availability. There is consequently a two-level scheduling: one at the Grid middleware level and the other one at the batch level. Neither the conception and validation of such algorithms nor their implementation are obvious. First, the execution of *large scale* experiments monopolizes the resources and cannot be reproduced. So it seems to be necessary to define common bases in order to simulate them and draw their profiles before trying to realize them in real life. Second, to efficiently schedule real life experiments, a Grid middleware must be able to get performance estimations on parallel resources. These have also to be used to dynamically tune parallel jobs in accordance with the parallel resource availability [12,6].

Contributions of this work mainly focus on the conception of an API which extends the functionalities of the Grid simulator Simgrid, allowing to easily simulate parallel resources and batch system in Grid computing. Realistic models of PBS [1] (or Torque [2]) and OAR [8] are built-in. The quality of the results obtained during the validation of this work allows us to use it as a simulation-based performance prediction tool embedded in the Grid middleware DIET.

2 Background

This section briefly describes our previous work, which has led to the design and contributions presented in this paper. Grid-TLSE [13] aims to provide an International Expert System for Sparse Linear Algebra relying on an international Grid computing environment which manages French and Japanese computing resources.

The architecture of Grid-TLSE [18] has been improved to the one [7] described in Figure 1. The architecture relies on the integration of a "protocol" interoperability between the French and Japanese middleware, respectively called DIET and AEGIS. DIET, developed by the GRAAL INRIA team project at LIP / ÉNS Lyon, is built upon the client/agent/server paradigm, and provides the GridRPC standard API [21]. This Grid middleware is able to find an appropriate server (running a DIET Server Daemon, SED), according to the information given in the client's request (*e.g.,* problem to be solved, size of the data involved), the performance of the target platform (*e.g.,* server load, available memory, communication performance) and the local availability of data stored during previous computations. Scheduling, which can be application specific, is distributed over a hierarchy of agents (Master and Local Agents). AEGIS (Atomic Energy Grid InfraStructure) is the next version of the IT Based Laboratory (ITBL) [14] middleware developed by the JAEA (Japan Atomic Energy Agency). In AEGIS, supercomputers are isolated from the Internet by a firewall for security reasons. Usually, a user of the AEGIS system connects to the computers through a Web portal, which has the accessibility for all the computers within AEGIS. Therefore, the portal equips file management, job submission, or the other basic functions for computation. On the other hand, AEGIS also proposes a control API to meet the expectations of advanced users.

The new architecture of Grid-TLSE, pictured in Figure 1, involves the following mechanisms which are very similar to the standard DIET operations: (a) After the

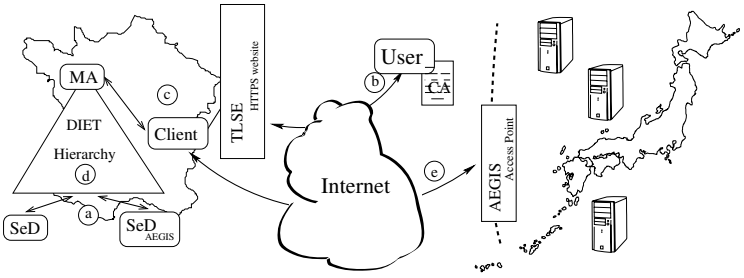

Fig. 1. Architecture of Grid-TLSE

deployment of all DIET components on Grid'5000 (including a specifically designed server daemon for the AEGIS system ($SeDs_{AEGIS}$), and composed of both binaries and configuration file(s)), each SED registers the services that it can solve to its agent in the hierarchy; (b) When a user performs a request through the secure Grid-TLSE Web portal, he must provide a certification file if he wants that his request can be executed on Japanese resources; (c) After being processed by the Grid-TLSE Web portal, the DIET client sends the corresponding request to the DIET hierarchy. The request is forwarded down (eventually pruned if the service is not available downward); (e) After having been sent back the identity of the host to contact, which can be managed either by the AEGIS middleware or by the DIET middleware, the common AEGIS-DIET client performs the call and the certificate may then be used. Once the problem solved, results are sent back to the client and so, made available to the user through the Grid-TLSE Web portal.

DIET is also able to obtain immediate information on parallel resources in order to perform cycle-stealing and online tunable parallel moldable job (with the possibility to set the number of processors to use at launch time) with as case of study, the sparse matrix solver PASTIX library [6]. Cycle-stealing policy was clearly dependent on the context of the work (analysis of a whole set of tasks without strict constraint on the experiment finishing date), and the work only used the immediate availability of the platform to tune the parallel jobs.

To improve these works, the DIET Grid middleware need estimations on when a job can be executed by a batch scheduler, which is dependent on the number of resources being requested. To be efficient, this number has to be chosen taking into account the jobs that may soon release reserved nodes which can benefit to the submitted parallel job (which would be launched later but with a smaller expected completion time). Furthermore, if a slot can be used depending on the batch scheduling policy (Conservative Backfilling for example), the Grid middleware may benefit of such information for its decisions.

Hence, Simbatch has been designed to be used as a simulation-based performance prediction tool to be used within DIET. Provided with information on the parallel system state, it takes into account the scheduler policy to instantly respond the different idle slots, with the number of processors that should be available as well as the duration of the idle slots.

3 Grid Simulators

There are numerous Grid simulators. Amongst them we can cite Bricks [24] for the simulation of client-server architectures; OptorSim [5], created for the study of scheduling algorithms dedicated to the migration and replication of data; GridSim [23] and Simgrid [19], which are by definition toolkits that provide core functionalities for the simulation of distributed applications in heterogeneous distributed environments.

Nonetheless, except for GridSim and Simgrid, systems are not generic [16,22]: they do not provide any API of reusable functions; moreover in an attempt to keep their study simple the employed scheduling policy is always *First Come First Served*; at last, they usually only implement sequential tasks, *e.g.,* they do not model *parallel tasks*.

If the GridSim toolkit covers several mandatory functionalities, it is hard to use the same code to at the same time address scheduling heuristic studies and performance predictions that can be used online to dynamically tune parallel applications according with the resource load. Furthermore, as it is written in JAVA, the use of GridSim, if feasible, is contradictory within the context of the lightweight deployment of the DIET Grid middleware.

Thus, we have chosen to integrate Simbatch [3] in the Simgrid toolkit to embed in DIET an efficient performance prediction tool, in order to improve its quality of service. In addition, its design also eases the conception and analysis of Grid scheduling heuristics.

4 The Simbatch API

Simbatch is a C API consisting of 2000 lines of code. It uses data types and functionalities provided by the Simgrid library to model clusters and batch systems. Simbatch provides a library containing already three scheduling algorithms [20]: *Round Robin* (RR), *First Come First Served* (FCFS) and *Conservative Backfilling* (CBF). The API is designed to easily let the user integrate its own algorithms. In order to visualize the algorithm behavior, a compliant output with the Pajé [4] software is available allowing the draw of the Gantt chart of the execution.

4.1 Modeling

Clusters consist of a frontal computer relied to interconnected computing resources following a specific topology. Resources of a cluster cannot be usually accessed directly from outside the cluster: communications must be done through the frontal. The batch manager system is run on the frontal node. Every jobs running on the nodes must have been submitted to the batch system. It receives requests from users, schedules them on the parallel resources and executes the corresponding task when needed. In this context, scheduling means that the batch scheduler must determine the starting time for each computing task and must allocate computing nodes for each of them. The computing tasks are generally parallel and could have both input and output data.

In Simbatch, a parallel task submitted by a client is modeled by the addition of different information to a Simgrid task data type such as the number of nodes, the

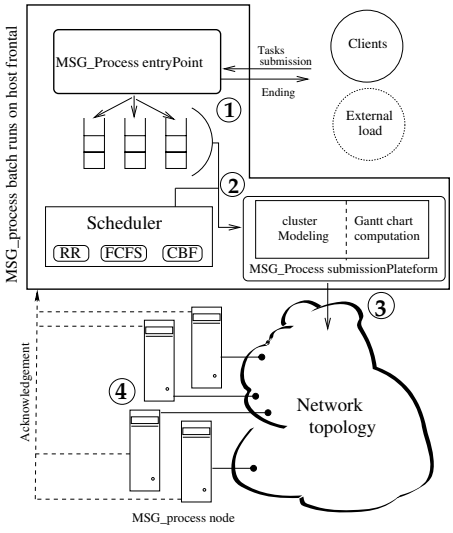

Fig. 2. Simbatch architecture

walltime, the run time. Other models are directly inherited from Simgrid. As exposed in Figure 2, the task treatment is made in the following manner:

1. The `entryPoint` process accept parallel tasks submission from the different clients and put them in the right priority queue.
2. Thanks to the modeling unit, the scheduler assigns the starting time to the tasks and reserves the computing resources needed for their execution. A global view of the cluster is obtained by calculating the Gantt chart.
3. The submission module manages the sending of each task at the starting date on the reserved resources. It controls the good respect of the reservation too. A task is killed if the walltime is exceeded.
4. Because Simbatch is build on top of Simgrid, it lets this one simulate communications and executions. When a task finishes its life cycle, an acknowledgement is sent to the `batch` process in order to update its global view of the cluster.

4.2 Using Simbatch

An experiment requires at least four files: a *platform file*, a *deployment file*, a *batch configuration file* and a file describing the tasks which will be submitted to each parallel resource, the *external load*.

Simgrid uses the *platform file* to describe resources which compose the simulated platform. It contains the description of all resources such as nodes, network links, the connectivity of the platform, etc.

Simbatch requires a *deployment file* in which the functions attached to the resources are defined. Thus, to define the use of a batch scheduler on frontal nodes, one must affect the `SB_batch` process provided by the Simbatch API to each frontal name.

Likewise, one must declare for each computing resource of each cluster the execution of the `SB_node` process provided by the Simbatch API.

The *batch configuration file* contains all information relative to each frontal node of the platform, like the number of waiting queues and the scheduling algorithm.

The *external load* is generated by the tasks submissions of the simulated Grid platform users. The file, whose name is recorded in the configuration file, describes the tasks specifications such as dates of submission, numbers of processors, walltimes, etc.

It is also possible to simulate an *internal load* for *each* batch scheduler. It aims at reproducing the submissions of tasks from clients who are directly connected to the parallel resource, *i.e.,* who are not actively participating to the simulated Grid platform. There is at most the same number of *internal load* files than the number of frontal nodes in the platform.

We give P.234 some of the files that we used for the experiments. The main code shows a client submitting a task to a batch scheduler described as follows: use of an external load, 3 priority queues, 5 nodes directly connected with a star topology and the CBF algorithm. In addition, we have also modeled the Grid'5000 node of Lyon [2].

5 Experimental Validation

Tasks generation. In order to validate the results obtained with Simbatch, we have built a workload generator using the GSL library [15]. It uses a Poisson's law with parameter $\mu = 300$ to generate inter-arrival time. Tasks specifications are determined by flat laws. Thus, CPU numbers are drawn from $U(1; 7)$, execution durations from $U(600; 1800)$ and walltimes are obtained by balancing the corresponding execution duration by a random number drawn from $U(1.1; 3)$.

Some experiments have been conducted with communicating tasks. They are all independent but require the communication of input data from the frontal node to one node allocated to the parallel task, and the communication of the output data back to the frontal node. In order to do this, we have created 6 files with a size of respectively $1, 2, 5, 10, 15, 20$ MB . One of this file is chosen randomly by a uniform law to be transfered as input data, while another file is chosen in the same manner to be transfered as output data.

Real experiments platform. OAR [8] is a batch reservation system developed by the project `MESCAL` in Grenoble. It is deployed on each site of the Grid'5000 platform. The scheduling algorithm used is CBF.

The 1.6 version of OAR has been installed on a cluster made of 1 frontal and 7 servers SuperMicro 6013PI equipped with a XEON processor at 2.4 GHz, each of them relied to a 100 Mbits/sec switch.

Protocol of experimentation. We have modeled the real platform by creating computing resources connected in a star topology. Then, thanks to our load generator, we have submitted the same load to both platforms (real and simulated). In this purpose, we have created a MPI computing task whose duration is given as parameter. The task

[2] `http://graal.ens-lyon.fr/simbatch`

is executed between two calls for time measuring in an OAR script. The precision of the time measure is about 1 second, so it is negligible compared to the task duration.

6 Results and Discussion

6.1 Validation of the Integrated Scheduling Algorithms

We show in Figure 3 the result obtained for one simple experience, described in Figure 1. One can see the Pajé Gantt chart on the top and beneath, the one obtained with the Drawgantt OAR. The tasks execution order is strictly the same (as it has always been, tested with a extensive set of experiments [17]): task 3, 4, 5 benefit from the *backfilling* and start their execution before task 2 which needs every nodes of the cluster. However, we can point out that Simbatch doesn't necessarily allocate the same nodes than OAR (task 3).

6.2 Accuracy of Simbatch Simulations

Two sets of experiments have been conducted with the second protocol: only computing tasks are involved in the first one, as the second one uses exclusively communicating tasks.

Experiments involving computing tasks. We have run numerous experiments for the first set of experiments, representing about 130 hours of computing on the cluster. Only computing tasks are involved. We present here a representative experiment for this set.

Table 1. Data used for experiment 1

Tasks	1	2	3	4	5
Processors number	1	5	2	1	3
Submission date	0	600	1800	3600	4200
Run time	10800	3300	5400	4000	2700
Reservation time	12000	4000	7000	5000	3500

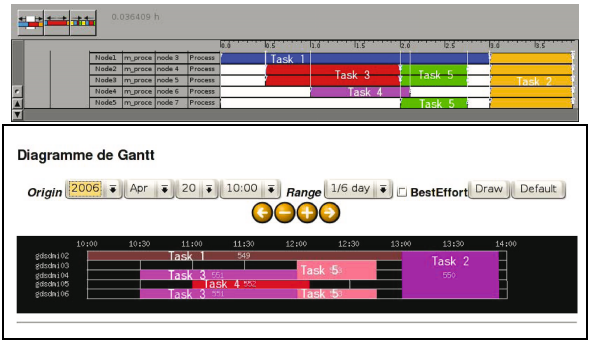

Fig. 3. Gantt chart for experiment 1: Simbatch (top), OAR (bottom)

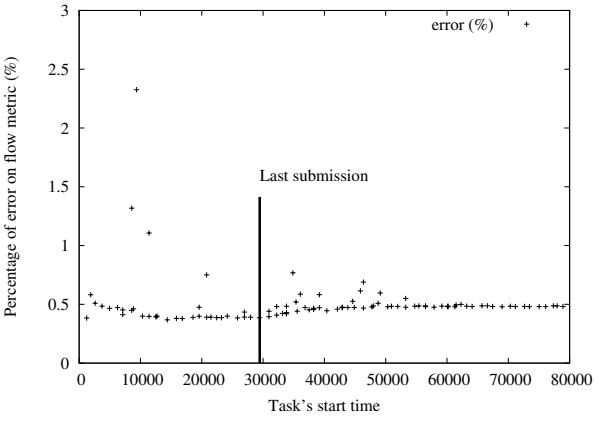

Fig. 4. Error ratios for an experiment scheduled with the Simbatch CBF and the real-life CBF implemented in OAR

The experiment consists in submitting the same set of 100 computing tasks to the real platform and to the simulated platform: results obtained with OAR was exactly 80392 seconds (about 22h) while the simulator gave us 80701 seconds. So the total execution time difference is only 308 seconds. It represents an error rate of 0.38%. This difference is mainly due to the mechanisms for interrogating the Mysql database, submitting tasks via ssh, etc., of OAR.

Figure 4 shows the error rate obtained for the flow metric in function of the tasks *execution* date. The flow of a task is the *time spent in the system*, *i.e.,* results from the addition of the waiting time passed in the queue, of the run time and of the communication costs. We can point out that the error rate is constant and generally below 1%, which is negligible compared to the precision of the measure. We can point out that for each experiment we have few tasks with an error rate above the 1%. This phenomenon is *not due to a scheduling error* due to some time precision here. In fact, some shorter and small tasks (time and processor) can enter the system and take advantage of the backfilling both with Simbatch and OAR. Because of the small gap between Simbatch and OAR starting time (thus between their ending time as well), the task begins a little later in reality, which can represent up to 15% and has only been observed once in our experiments (the second maximum observed is 6%).

An arrow is also drawn at time 29454: it represents the date of the last *submission*. In a dynamic environment, if we give to Simbatch *every specification* of a set of tasks submitted to a batch scheduler, then Simbatch should be able to make a reliable prediction on the execution of this set.

Experiments involving communicating tasks. Since we obtained excellent results for the simulation of batch scheduler for parallel tasks without communication costs, we have decided to go further and we have tested Simbatch with experiments involving communication costs. Hence, we transfer some data from the frontal node to one of the allocated nodes for the parallel task. Once the computation done, we transfer back some data from the same allocated node to the frontal.

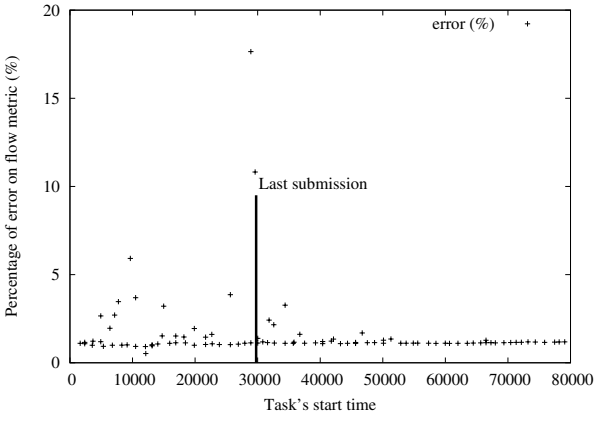

Fig. 5. Error ratios for an experiment involving communication costs scheduled with the Simbatch CBF and the real-life CBF implemented in OAR

We have run the experiments on our platform with OAR and in simulation with Simbatch. Then, we have measured the flow for each task and we have calculated the error on the flow metric between simulation and real experiments.

Figure 5 depicts one representative experiment. The error rate is low, with an average around 2%. However a few tasks have a higher error percentage. After having analysed our results, we can point out that those tasks have the same profile, *i.e.,* small computation time and few resources needed (typically 1 processor). When we analyse deeply, we can conclude that those tasks are taking advantage of the backfilling in simulation contrarily to the reality. In spite of the fact that some tasks are scheduled earlier in simulation than in reality, the impact is very small on the other task flow: the duration of the task which benefited from a backfilling represents a small percentage of other task flow.

Thus, Simbatch obtains realistic results for simulated experimental studies. It allows to easily model parallel resources managed by a batch scheduler. The good quality of its simulation shows its relevancy in the study of scheduling algorithms for the Grid. Furthermore, its use is straightforward as prediction module in a Grid middleware.

7 Conclusions and Future Works

The submission of parallel tasks by a Grid middleware is not straightforward, particularly due to the lack of profiling functionalities in batch schedulers. Nonetheless, performance estimations must be used to both efficiently schedule tasks on the Grid and tune accordingly parallel tasks with the parallel resource load and scheduling policy.

As a step in the Grid-TLSE architecture, we have designed Simbatch. It can be embedded in a Grid middleware to give accurate predictions. We have detailed those functionalities and we have specified the models we use. Then we have shown the facility for every Simgrid user to use Simbatch thanks to the examples coming from our validation work.

The main scheduling algorithms (*Round Robin*, *First Come First Served* and *Conservative BackFilling*; the last two are respectively implemented in PBS, and in MAUI and OAR) are integrated and have been validated by several simulation experiments. Moreover, we have compared results obtained from Simbatch simulations with the ones from the real-life batch scheduler OAR. Simbatch shows very good precision with an error rate in general less than 2%.

There are numerous perspectives. Among them, we want to test and eventually extend Simbatch to batch schedulers like SGE or Loadleveler; we want also to design scheduling heuristics to take advantage of such predictions and integrate them in DIET for immediate use in the Grid-TLSE system.

References

1. http://www.openpbs.org/
2. http://old.clusterresources.com/products/torque/
3. http://simgrid.gforge.inria.fr/doc/contrib.html
4. http://www-id.imag.fr/Logiciels/paje/
5. Bell, W., Cameron, D., Capozza, L., Millar, P., Stockinger, K., Zini, F.: Optorsim - a grid simulator for studying dynamic data replication strategies. Journal of High Performance Computing Applications 17 (2003)
6. Caniou, Y., Gay, J.-S., Ramet, P.: Tunable parallel experiments in a gridrpc framework: application to linear solvers. In: VECPAR 2008 International Meeting on High Performance Computing for Computational Science (2008) (to appear)
7. Caniou, Y., Kushida, N., Teshima, N.: Implementing interoperability between the AEGIS and DIET GridRPC middleware to build an International Sparse Linear Algebra Expert System. In: Second International Conference on Advanced Engineering Computing and Applications in Sciences, ADVCOMP 2008 (2008)
8. Capit, N., Da Costa, G., Georgiou, Y., Huard, G., Martin, C., Mounier, G., Neyron, P., Richard, O.: A batch scheduler with high level components. In: Cluster computing and Grid 2005, CCGrid 2005 (2005)
9. Cappello, F., Desprez, F., Dayde, M., Jeannot, E., Jegou, Y., Lanteri, S., Melab, N., Namyst, R., Primet, P., Richard, O., Caron, E., Leduc, J., Mornet, G.: 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)
10. Caron, E., Desprez, F., Fleury, E., Lombard, F., Nicod, J.-M., Quinson, M., Suter, F.: Une approche hiérarchique des serveurs de calculs. In: Calcul réparti à grande échelle, Hermés Science, Paris (2002)
11. Casanova, H., Dongarra, J.: Netsolve: A network server for solving computational science problems. In: Proceedings of Super-Computing, Pittsburg (1996)
12. Cirne, W., Berman, F.: Using moldability to improve the performance of supercomputer jobs. J. Parallel Distrib. Comput. 62(10), 1571–1601 (2002)
13. Daydé, M., Desprez, F., Hurault, A., Pantel, M.: On deploying scientific software within the Grid-TLSE project. Computing Letters 1(3), 85–92 (2005)
14. Fukuda, M.: ITBL – toward constructing a new R & D environment, vol. 55, pp. 19–23 (2002)
15. Galassi, M., Theiler, J.: The Gnu Standard Library (1996)
16. Garonne, V.: DIRAC - Distributed Infrastructure with Remote Agent Control. Ph.D thesis, Université de Méditéranée, Décember (2005)

17. Gay, J.-S., Caniou, Y.: Simbatch: an api for simulating and predicting the performance of parallel resources and batch systems. Technical Report RR2006-32, LIP ENS-Lyon, Université Claude Bernard Lyon 1, Lyon, (October 2006)
18. Kushida, N., Suzuki, Y., Teshima, N., Nakajima, N., Caniou, Y., Daydé, M., Ramet, P.: Toward an International Sparse Linear Algebra Expert System by interconnecting the ITBL computational Grid with the Grid-TLSE platform. In: VECPAR 2008 International Meeting on High Performance Computing for Computational Science (2008) (to appear)
19. Legrand, A., Marchal, L., Casanova, H.: Scheduling distributed applications: the simgrid simulation framework. In: IEEE Computer Society (ed.) 3rd International Symposium on Cluster Computing and the Grid, p. 138. IEEE Computer Society, Los Alamitos (2003)
20. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. In: IEEE (ed.) IEEE Trans. Parallel and Distributed Systeme, pp. 529–543. IEEE, Los Alamitos (2001)
21. Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C., Casanova, H.: A GridRPC Model and API for End-User Applications (December 2003)
22. Ranganathan, K., Foster, I.: Decoupling computation and data scheduling in distributed data-intensive applications. In: 11th IEEE International Symposium on High Performance Distributed Computing, HPDC-11 (2002)
23. Sulistio, A., Cibej, U., Venugopal, S., Robic, B., Buyya, R.: A toolkit for modelling and simulating data grids: An extension to gridsim (accepted December 3, 2007) (in press)
24. Takefusa, A., Casanova, H., Matsuhoka, S., Berman, F.: A study of deadline for client-server sytems on the computational grid. In: 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), pp. 406–415 (2001)

Annexe A

```
<?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>
```

Deployment file

```
<?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="98.0949999999999999"/>
  <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>
```

Platform description file

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

<config>
        <!— Global settings for the simulation —>
        <global>
                <file type="platform">platform.xml</file>
                <file type="deployment">deployment.xml</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 of the simulated batch system

```
#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;
}
```

Simgrid main code