iSPD: an iconic-based modeling simulator for distributed grids

Aleardo Manacero, Renata S. Lobato, Paulo H.M.A. Oliveira, Marco A.B.A. Garcia,
Aldo I. Guerra, Victor Aoqui, Denison Menezes, Diogo T. da Silva
Univ. Estadual Paulista - UNESP
Computer Science and Statistics Dept.
(aleardo,renata)@ibilce.unesp.br

Keywords: iconic modeling, grid simulation, distributed systems

Abstract

Simulation of large and complex systems, such as computing grids, is a difficult task. Current simulators, despite providing accurate results, are significantly hard to use. They usually demand a strong knowledge of programming, what is not a standard pattern in today's users of grids and high performance computing. The need for computer expertise prevents these users from simulating how the environment will respond to their applications, what may imply in large loss of efficiency, wasting precious computational resources. In this paper we introduce iSPD, **i**conic **S**imulator of **P**arallel and **D**istributed Systems, which is a simulator where grid models are produced through an iconic interface. We describe the simulator and its intermediate model languages. Results presented here provide an insight in its easy-of-use and accuracy.

1. INTRODUCTION

The use of high performance computing is growing remarkably in this century. It is not restricted to high-end scientific applications anymore. This growth has been enabled by the introduction of less expensive systems associated with the use of resource sharing systems, such as computer grids. As a result of the improved demand a large amount of HPC users that are neither computer experts nor have a large computer support team to help with system analysis has appeared. These new users create a new demand to HPC developers, which is the creation of easy-to-use tools for system's evaluation, since they need to grasp the best performance of an environment that they do not fully understand.

Among the tools for system's evaluation there are a wide range of simulators. Simulation is considered an important approach for performance evaluation since it can provide modeling flexibility with reasonable accuracy. It is also relatively cheaper than actual benchmarking and can be used in any stage of a system development. The major drawback with simulation is that most of simulators demand the knowledge of specific programming languages, either simulation or conventional languages. In the case of grid simulators this is totally true, with the most influential tools, namely Simgrid [4] and Gridsim [3], being strongly dependent of scripting languages.

In this paper we describe an iconic based simulator, iSPD (*i*conic *S*imulator of *P*arallel and *D*istributed Systems), which allows the creation of grid models through a graphical environment. The simulator transforms the graphical model into a queue system, which is executed to provide performance data for the user. The whole process can be performed very intuitively, starting from the iconic model of the grid that needs to be simulated, including its resources and tasks that must be executed.

In the following pages we firstly contextualize iSPD among the other grid simulators available, then describe its specification and design. Results from its use are also provided, alongside the conclusions drawn from these tests.

2. RELATED WORK

Grid simulators are in use for several years now. Some different simulators have been proposed, although only a few are regularly used. During this section a brief description of them is provided. A greater attention will be provided to the mainstream simulators, Simgrid and Gridsim, since they are well maintained and developed.

Simgrid [5] is the first simulator proposed and still one of the most used. Its initial goal was the evaluation of centralized scheduling policies for heterogeneous and distributed computational environments. New versions of Simgrid have been released continually, although it still lacks an easy-to-use interface to create models. One of its strenghts is the capability to model background traffic and computation, which is not present in most simulators. Models are configured by XML and C files, which is not easy for non-expert users. Currently there is a GUI, Grid Matrix [12], that allows for an easy control of the simulation process, besides the schedulers continue to be provided through their actual codification in C.

GridSim [3] is another largely used simulator, currently in its version 5.0. It allows for modeling of different classes of environments, including schedulers and machines. It is based in the SimJava simulation engine. It is quite flexible and has an interface that makes easier to model some types of computing grids. It is more portable than other simulators since is

Java-based. Models are built through a Java program, using pre-build classes for tasks and other components.

GangSim [6] was developed to evaluate scheduling policies in grid environments. It allows the analysis of interactions between local and global schedulers. This feature is very interesting and it is under development in iSPD. Gangsim, as the other simulators, does not enable an easy modeling interface, demanding the writing of scripts in an internal language.

OptorSim [2] was initially developed to evaluate dynamic replication algorithms used to optimize data location over the grid. Optorsim has been used in this field, which is a major difference when compared to the usual application of grid simulators in scheduling analysis. Compared with iSPD it does not have a simple interface to model grids while iSPD is not currently capable to manage data replication.

BeoSim [8, 9] is a discrete event simulator aimed to computer grids assembled as Beowulf clusters, interconnected through a dedicated network. It enables the evaluation of smaller grids under different workloads and scheduling policies.

GSSIM (Grid Scheduling Simulator) [7, 10] was built over GridSim aiming to solve the problems with workload generation and scheduling levels present in other simulators.

These simulators have been used mostly to evaluate scheduling policies. In order to perform such evaluation, it is necessary to model the grid (hosts and networks), the workload, and the scheduling policies themselves. These tasks are not easy to perform in the simulators just described. iSPD, however, makes all these tasks easier to be performed while providing comparable accuracy. Hosts and their connections are easily modeled by an iconic interface, while the scheduling policies are parameters provided to specific components and may also be modeled through a simple interface.

3. THE iSPD

The design of iSPD is structured in three basic modules: the graphical interface, the language interpreter/decoder and the simulation engine. Besides these modules, some additional interfaces are present, mostly concerned with importing/exporting models and metrics collection. The interaction between modules can be seen in Figure 1. In that Figure one sees that the user inputs the model through an iconic model, which is translated to a simulatable model, receiving the results from its simulation. The simulation can make use of a workload database or a random workload. There is also the possibility to convert external models to an iconic model or vice versa[1].

[1]Currently it is possible to convert Simgrid models to iSPD models

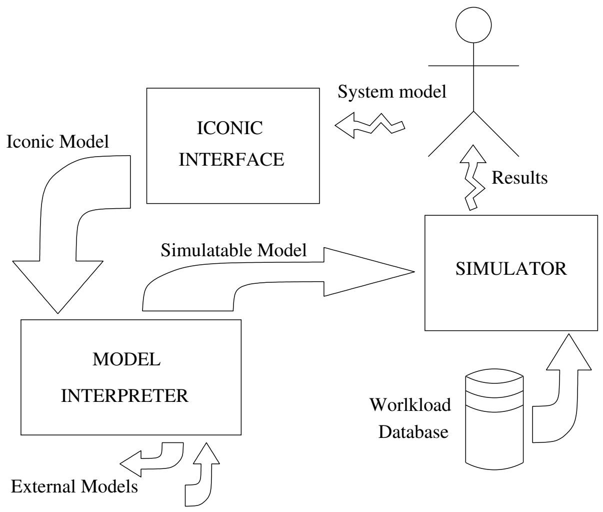
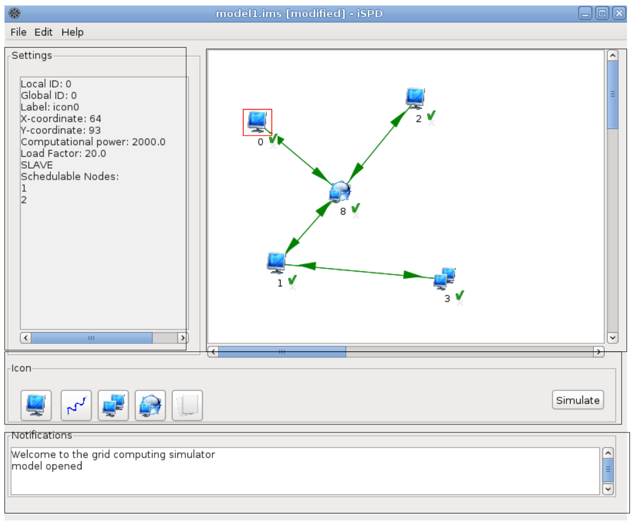


Figure 1. iSPD Basic Structure

Grids can be simulated through some basic components, such as clusters, communication links, and individual hosts. A basic model can be depicted in Figure 2, where the model contains three individual nodes (nodes 0, 1 and 2), a cluster (node 3), a pointo-to-point communication link (between nodes 1 and 3) plus internet connections (internet cloud is node 8). From this figure it is also possible to identify the components in the interface. The main component is the drawing sector, where the user actually draws the model to be simulated. On its side there is an information display, the “Settings” window, which shows the parameters of a selected icon (node 0 in that case). Just below these windows there are the icons menu and the “Simulate” button. Finally there is the “Notifications” window, where a log of the session is registered.

Once the model is entered to the system, the interface generates a text file containing its description, using a block-based iconic description language. This language describes a grid through a set of individual description blocks using a specific grammar for this. Figure 3 shows the file created for the model drawn in Figure 2. This file is then interpreted by the model converter, leading to a second text file containing the grid described as a queue model using a queueing description language 4.

Although the use of two different languages may seem clumsy, it saves a lot of effort when dealing with imported models. Using this approach it is possible to restrict the conversion to/from other simulators to the iconic language, enabling the user to easily recover models built for those simulators to the iSPD iconic interface. Using a separate language to describe the simulation constraints and model also avoids the need for queue-oriented parameters or objects in the iconic interface. Figure 4 shows part of the queueing model for the given example, where some lines were omitted for the sake



```
Local ID: 0
Global ID: 0
Label: icon0
X-coordinate: 64
Y-coordinate: 93
Computational power: 2000.0
Load Factor: 20.0
SLAVE
Schedulable Nodes:
1
2
```

```
Welcome to the grid computing simulator
model opened
```

Figure 2. iSPD Basic Structure

```
HOST icon1 1200.000 30.000 MASTER RoundRobin LMAQ icon3
HOST icon2 2000.000 30.000 SLAVE
HOST icon0 2000.000 20.000 SLAVE
CLUSTER icon3 8 20000.000 10000.000 0.400000 RoundRobin
INET icon111 20.000 1.000000 40.000
LINK lan7 10000.000 0.400000 10.000 CONNECTS icon3 icon1
LINK lan10 10000.000 0.400000 10.000 CONNECTS icon2 icon111
LINK lan5 10000.000 0.400000 10.000 CONNECTS icon1 icon3
LINK icon14 10000.000 0.400000 10.000 CONNECTS icon1 icon111
LINK lan12 10000.000 0.400000 10.000 CONNECTS icon111 icon0
LINK lan11 10000.000 0.400000 10.000 CONNECTS icon111 icon1
LINK lan13 10000.000 0.400000 10.000 CONNECTS icon111 icon2
LINK lan9 10000.000 0.400000 10.000 CONNECTS icon0 icon111
LOAD RANDOM
11 28 55 0.98
11 16 22 1.0
0 2 30
```

Figure 3. An example of iSPD's iconic description language

of simplicity.

The process continues with the simulation engine initially converting the simulatable model to a queueing network. Once the network is created, iSPD starts the model simulation using scheduling policies provided as grid parameters. Results from the simulation are then aggregated in order to provide a series of performance metrics for the user. The file containing the results is very large and will not be shown here. It basically contains the whole simulation history, from which performance parameters are extracted. The performance metrics currently provided by iSPD include:

- Average turnaround times, as the average time spent to complete each task;
- System's efficiency, as a measure of system demand and occupation;

```
MODEL
TASK
RANDOM 11 28 55 0.98
11 16 22 1.0
0 2 30
END_TASK
SERVICE_CENTERS
CS_0 cs_icon1 1 1 QUEUES q_icon1 SERVERS
serv_icon1 0 1200.000 30.000 MASTER RoundRobin LMAQ cs_icon3
CS_2 cs_lan13 1 1 QUEUES q_lan13 SERVERS
serv_lan13 1 10000.000 10.000 0.400000
      .
      .
      .
CS_1 cs_icon3 2 8 RoundRobin QUEUES q_0_icon3 q_1_icon3 SERVERS
serv_icon3 0 20000.000 10000.000 0.400000
CS_2 cs_lan10 1 1 QUEUES q_lan10 SERVERS
serv_lan10 1 10000.000 10.000 0.400000
END_SERVICE_CENTERS
CONNECTIONS
cs_icon111     cs_lan13
      .
      .
      .
cs_icon2       cs_lan10
cs_lan10       cs_icon111
END_CONNECTIONS
END_MODEL
```

Figure 4. An example of iSPD's queueing description language

- User's satisfaction, as a measure of fairness in user's attention; and
- Average waiting times, as a measure of contention for resources.

In order to correctly model a grid environment through a queueing network it was necessary to map each resource type to a network configuration. This resulted in modeling communication links and single hosts as 1-queue-1-server networks, computer clusters as 1-queue-N-servers, and the whole grid as N-queues-N-servers. In fact, a grid can be seen as a recursive assembly of clusters, single nodes and communication links.

Although the user may configure the probability distribution functions used by the service centers in the queueing network, there are few distributions that must be present as defaults. Therefore, iSPD offers random generators for the Poisson, exponential and two-stage uniform functions. The choice for these functions is based on previous results provided by Lublin and Feitelson [11], showing that they make good models for distributed systems.

3.1. Iconic and Queueing languages

The core of iSPD is the use of two languages to represent the iconic, graphical in nature, model, and the simulatable model. The grammars for these languages are reasonably simple, generated using context-free grammars that include all needed objects to create those models. Here we provide only the main parts of the grammars that generate the models.

The grammar of the iconic modeling language is context-free, with a reasonably small amount of symbols. The reserved words in this language include “HOST”, “CLUSTER”, “LINK”, “INET”, “MASTER”, “SLAVE”, and few other internally defined terms for input parameters. The starting token for a model specification is $<model>$. It leads to a list of the available icons from the user interface. The rules in Figure 5 describe the model global specification and the HOST grammar rules. The grammar rules for the remaining components of a model, which are similar to the rules for HOST, will be ommited here.

```
<model> ::= <icons>
<icons> ::= {<icon>}+
<icon> ::= <node> | <cluster> | <link> | <inet> | <load>
<node> ::= HOST <server_ID> <computing_power> <load_avg> <nodetype>
<nodetype> ::= MASTER <clusteralg> HOST_LIST <slaves> | SLAVE
<server_ID> ::= <identifier>
<computing_power> ::= <real>
<load_avg> ::= <real>
<clusteralg> ::= RR | WORKQUEUE | FPLTF
<slaves> ::= {<server_ID>}+
```

Figure 5. Section of the iconic description language grammar

The first three lines in figure 5 define the icons that are present in the modeling interface. The fourth line is the starting point for the definition of a single host in the model. As it can be seen, the icon for a host is described by rules defining the node type (master or slave), processing capacity (computing power) in MFlops, and load average (percentage). If the node is a master it also defines the scheduling algorithm and the list of slaves attached to it.

Using the same approach, the grammar of the queue modeling language is also context-free and reasonably simple. Figure 6 shows the top levels of the grammar for the queue modeling language. Line 1 shows the starting symbol for this grammar ($<queue\_model>$), while the remaining lines describe the starting ramifications for the rules defining service centers and the connections between them.

```
<queue_model> ::= MODEL <definitions_list>  END_MODEL
<definitions_list> ::= <definitions_list> <definitions> | <definitions>
<definitions> ::= <define_SC> | <connections> | <simulation_definitions>
<connections> ::= CONNECTIONS <connection_list> END_ CONNECTIONS
<connection_list> ::= <SC_ID> <SC_ID> | <connection_list> <SC_ID> <SC_ID>
<define_SC> ::= SERVICE_CENTERS <SC_list> END_SERVICE_CENTERS
```

Figure 6. Initial section of the queue description language grammar

Figure 7 shows the initial rules for service centers and for servers. Lines 4 to 7 in this figure (`<SC_i>`) define rules for specific service centers, which are responsible to simulate different grid elements. SC0 is mapped to processing nodes, demanding information about processing speed and occupation for example. SC1 is mapped to clusters, demanding information about local bandwidth, processing speed, occupation and scheduling policy. SC2 is mapped to communication links and SC3 to the internet cloud.

```
<define_SC> ::= SERVICE_CENTERS <SC_list> END_SERVICE_CENTERS
<SC_list> ::= <SC_list> <SC> | <SC>
<SC> ::= <SC_0> | <SC_1> | <SC_2> | <SC_3>
<SC_0> ::= SC_0 <SC_ID> <prmtrs> QUEUES <queue> SERVERS <server1>
<SC_1> ::= SC_1 <SC_ID> <prmtrs> <policy> QUEUES <queues>
                                                        SERVERS <server2>
<SC_2> ::= SC_2 <SC_ID> <prmtrs> QUEUES <queue> SERVERS <server3>
<SC_3> ::= SC_3 <SC_ID> <prmtrs> QUEUES <queue> SERVERS <server3>
<prmtrs> ::= <num_queues> <num_servers>
<server1> ::= <server_ID> <server_type> <proc_capacity> <occup_rate> <master>
<server2> ::= <server_ID> <server_type> <proc_capacity> <bandwidth> <latency>
<server3> ::= <server_ID> <server_type> <bandwidth> <occup_rate> <latency>
```

Figure 7. Description of service centers invf the queue description language grammar

3.2. Simulation engine

The simulation engine reads the system’s model, after its conversion from the iconic to the queueing model and simulates it through an event-based process. It is composed by two modules: one that manages the service centers, including queues and their respective servers, and other that manages the scheduling policies used by each server. The different events managed by the simulation engine are:

- **Task arrival:** when a specific task is added to a server’s queue;
- **Task service:** when a task is served by the server;
- **Task delivery:** when a task is removed from a service center and, eventually, creates a new **Task arrival** in another center.

Figure 8 shows how the simulator works. It follows the Event Scheduling/Time Advance [1] model, managing the future events (FE) list through correct insertions and removals, and adjusting the simulated time. When no more events are present in the list, the engine produces all relevant metrics that may have interest for the grid evaluation.

Each element from the grid model is mapped to specific servers (service centers) in the queue model. Specific centers include:

- **Communication service centers**
  - **Direct link centers:** following the one queue-one server model, with a FIFO service policy. These centers are used to connect two other centers in order to move data around the grid;

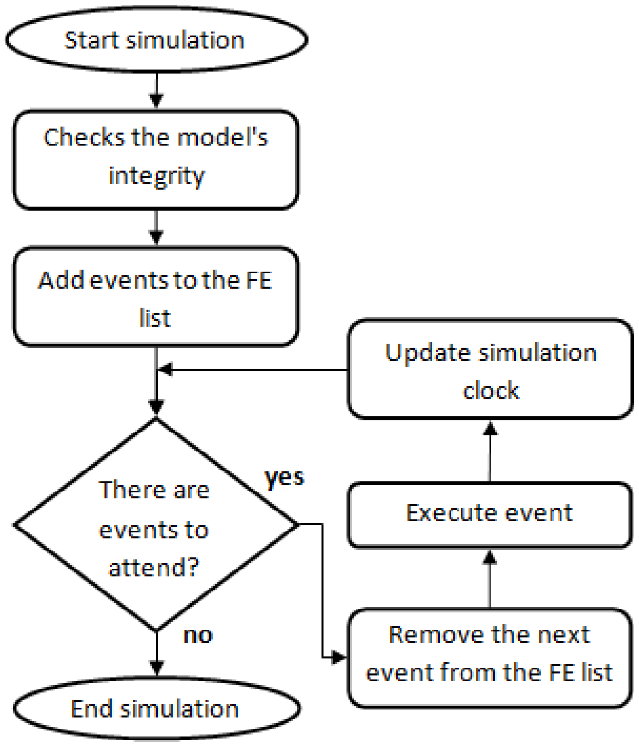

Start simulation
Checks the model's integrity
Add events to the FE list
Update simulation clock
There are events to attend?
yes
no
Execute event
Remove the next event from the FE list
End simulation

Figure 8. Simulation Process

- **Switching centers:** following the multiple queues-one server model, with a FIFO service policy. They are used to connect cluster nodes;
- **Internet center:** following a one queue-multiple servers model, where the number of servers is allowed to grow indefinitely. This allows to emulate a zero length queue that is fed by and feeds other communication service centers;

• **Processing service centers**

- **Host service centers:** follow a one queue-multiple servers model, with a FIFO policy. This type of center emulates the jobs processing, containing one or more servers to represent hosts with single- or multiprocessors with shared memory;
- **Centralized server service centers:** follow a one queue-multiple servers model, with FIFO policy. It differs from the host centers by being the center that executes the global job scheduling, directing the events to the servers that will actually execute the job.

4. iSPD EVALUATION

Since its goal is to provide an intuitive modeling interface there are two different lines of test to be presented. Firstly, the simulator has to be accurate, that is, the predicted behavior of a set of tasks in a given grid environment must be reproduced during simulation. Secondly, it is needed to compare how easy a model can be configured using iSPD against other simulators. These results are presented now.

4.1. Accuracy

In order to verify iSPD accuracy several different tests were performed. A first set of tests aimed to verify its correctness against simple queue models. A second set of tests aimed to verify its correctness against complete grid models. Results from the first set will not presented here, since they were performed only to verify if the elementary components of iSPD were implemented correctly and to tune up the random generators used in the simulation engine. The latter set of tests was composed by simulations using iSPD and Simgrid.

The tests involved a real program running on a research cluster, named cluster-GSPD, and comparing its measured execution times with simulated times from iSPD and Simgrid. The cluster runs Debian linux, version 2.6.26, and is composed by a front-end plus eight nodes of pentium dual machines, with 2 Gbytes of RAM. Figure 9 presents a schematics of how the cluster is structured. The actual processing speed of each node of this cluster was measured and resulted, in average, 700 billion instructions per second (700,000 Mflops). This value was used by the simulator as the average computational speed for the system.

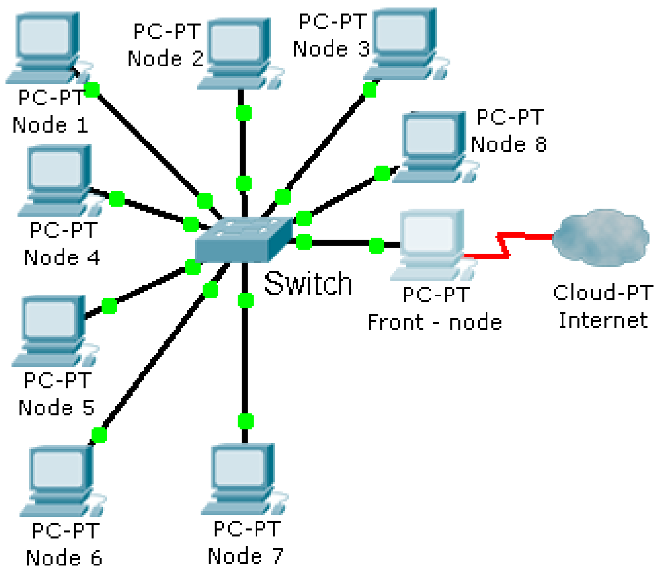


Figure 9. Cluster-GSPD schematics

A round-robin policy was implemented in the cluster by a MPI program. It is composed by a master process, running in the front-end, and eight slaves, one in each of the cluster's nodes. The master process creates tasks and distributes them following the round-robin policy. Each slave has three threads with specific functions: receiving data, munching data and sending results.

Models for such environment were created both in iSPD and Simgrid. For this test each task has a computing cost of 384.45 Mflops and a communication cost (data transferred) of 1 kbits. Tests were performed with the number of tasks ranging from 20 to 120, in 20 tasks steps. The plot in Figure 10 presents the average measured execution times for the complete job in the cluster-GSPD and the simulated execution times achieved with Simgrid and iSPD.

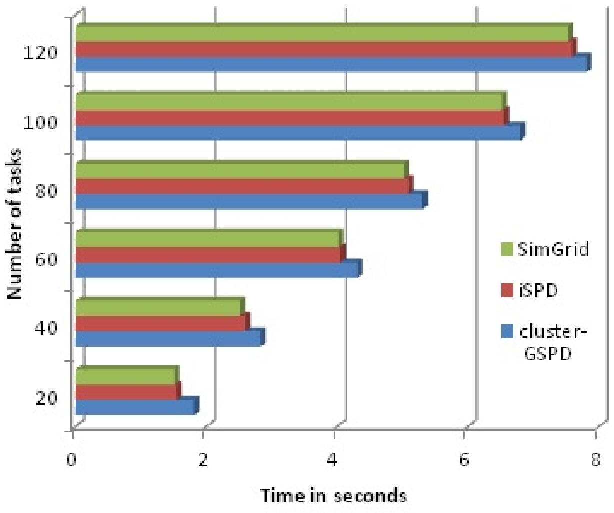


Figure 10. Measured and simulated times for different number of executed tasks (Round-Robin)

As one can see from Figure 10, the results provided by iSPD are very close to those provided by Simgrid, with their difference averaging 1.6%. When compared to the actual measurements from the cluster-GSPD, iSPD performed reasonably well too. The average error was 6.6% (8.0% for Simgrid), and the error linearly decreased when the number of tasks increased (2.8% for 120 tasks). These results, also summarized in Table 1, are a strong indicator that the iSPD generated models are quite accurate and can easily map real environments.

4.2. Easiness of use

Since the major claim made about iSPD is that it makes the modeling process easier than other grid simulators, it is necessary to provide proof for that. Fortunately, this can be made simply by comparing the processes for creating the same model in iSPD and in one of the available simulators. During the previous section this task has already been performed to evaluate iSPD's accuracy. Therefore, that modeling process will be reported here.

Table 1. Simulation results for models using the Round-Robin scheduler (column "Time" refers to the simulated time (iSPD/Simgrid) or the measured time (cluster-GSPD).

system	# of tasks	Time (sec)	% of cluster
cluster-GSPD	20	1.819	–
iSPD		1.552	85.32
Simgrid		1.511	83.07
cluster-GSPD	40	2.831	–
iSPD		2.592	91.54
Simgrid		2.513	88.75
cluster-GSPD	60	4.306	–
iSPD		4.050	94.06
Simgrid		4.016	93.27
cluster-GSPD	80	5.308	–
iSPD		5.090	95.90
Simgrid		5.018	94.53
cluster-GSPD	100	6.797	–
iSPD		6.550	96.37
Simgrid		6.521	95.94
cluster-GSPD	120	7.807	–
iSPD		7.590	97.23
Simgrid		7.523	96.37

To create the model using iSPD it was necessary only to insert the cluster icon and insert its parameters. Since the round-robin scheduler is already available, no other activity had to be performed by the user. The information that had to be provided consisted only of the number of nodes, scheduling policy, processing speed, communication speed and system load, in the grid side, and computational demand in the task side. All of these data is necessary for any simulation model, which is not different in iSPD's case.

On the other hand, to create the Simgrid model it was necessary to write a program for the round-robin scheduler (although it has just 200 lines of C code, it demands knowledge about C and the structures of the scheduling algorithm) and two XML definitions file. Besides the XML files, describing the environment and the application, are straightforward, they can be quite long in order to represent individual links, nodes and so on. For the model in review, the application file has 23 lines and the environment file has more than 120 lines. Figure 11 shows part of the environment file for this model. In that figure it is possible to identify three sections. One to define the processing nodes, one to define the communication links, and a final one to define the routing adopted for these links.

Despite not shown here, it is obvious that a C program with 200 lines is not as simple as simply applying a pre-defined scheduling policy, as is the case for iSPD. Therefore, it is possible to assure that it is easier to model grid systems using iSPD than using Simgrid. Since all other grid simulators in

```
<?xml version='1.0'?>
<!DOCTYPE platform_description SYSTEM "surfxml.dtd">
<platform_description version="1">
  <!-- computing power - Mflop/s   bandwidth - Mb/s  latency s -->
  <cpu name="gspd-fe" power="768.9"/>
  <cpu name="gspd-node1" power="768.9"/>
     ...
  <cpu name="gspd-node7" power="768.9"/>
  <cpu name="gspd-node8" power="768.9"/>

  <!-- connections between front-end and internal nodes -->
  <network_link name="lan" bandwidth="100" latency="0.001"/>
  <network_link name="lan1" bandwidth="100" latency="0.001"/>
  <network_link name="lan2" bandwidth="100" latency="0.001"/>
     ...
  <network_link name="lan35" bandwidth="100" latency="0.001"/>
  <network_link name="lan36" bandwidth="100" latency="0.001"/>

  <!-- routing -->
  <route src="gspd-fe" dst="gspd-node1"><route_element name="lan1"/></route>
  <route src="gspd-node1" dst="gspd-fe"><route_element name="lan1"/></route>
  <route src="gspd-fe" dst="gspd-node2"><route_element name="lan2"/></route>
     ...
  <route src="gspd-node7" dst="gspd-node8"><route_element name="lan36"/></route>
  <route src="gspd-node8" dst="gspd-node7"><route_element name="lan36"/></route>
</platform_description>
```

Figure 11. Partial view of XML file describing the environment for Simgrid

use have modeling characteristics similar to Simgrid, it is also possible to assume that it is easier to model grid systems using iSPD.

As a side note, using a GUI such as Grid Matrix with Simgrid [12], solves only the creation of the XML files. with Grid Matrix the user can draw the grid topology and insert information about the environment and the application using its GUI. The scheduling policy, however, still have to be programmed from the scratch, what means a 200 lines C code for the Round-Robin scheduler, for example.

5. CONCLUSIONS

The work presented in this paper is concerned with a new approach for grid simulation. Its goal was to provide an easy-to-use modeling interface, which enabled non-expert users to model grid systems without the need to write complex programs or scripts. This easier modeling should come without prejudice in the simulator's accuracy.

The tool built under these specifications, iSPD, showed both accuracy and simplicity. As the results presented in the previous section showed, iSPD is quite accurate, providing simulation results very close to results from real executions. It is also simple to use, since its graphical interface allows for an easy modeling process, based on icons and specific parameters asked by the interface. All of this at a very low processing cost, providing very reasonable simulation times even in a simple desktop computer.

New developments in iSPD are already under way. They include the addition of more interpreters for models from/to other simulators, where current works are in interpreters from iSPD to Simgrid and to/from Gridsim models. In another front, it is been developed an interface to automaticaly build grid schedulers. With such interface it will be possible to define and evaluate new scheduling policies with simplicity. In this interface an user will provide parameterized rules for the policy and an intermediate Java class will be generated to emulate such policy, which could be added to the scheduler library offered by iSPD. Other additions under consideration include the simulation of virtualized environments, and computing clouds.

To finish this section we can conclude that iSPD is an interesting option to model grids. It is accurate and easy to use. It has an intuitive interface based on iconic modeling, which distinguished it from other grid simulators in use.

ACKNOWLEDGMENTS

The authors must acknowledge to FAPESP, that supported this research through grants 08/09312-7 and individual scholarships. We also want to acknowledge G.C. Furlanetto and R.S. Stabile, for their contributions in the execution of the benchmarks.

REFERENCES

[1] J. Banks, J. S. Carson, D. M. Nicol, and B. L. Nelson. *Discrete-Event System Simulation.* Prentice-Hall, 3nd edition, 2001.

[2] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockingger, and F. Zini. Simulation of dynamic grid replication strategies in optorsim. In *Proc. of the ACM/IEEE Workshop on Grid Computing*. Springer-Verlag, 2002.

[3] R. Buyya and M. Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Pract. and Exper.*, 14:1175–1220, 2002.

[4] H. Casanova, L. Legrand, and A. Marchal. Scheduling distributed applications: The simgrid simulation framework. In *Proc. of the 3rd IEEE Intl Symp. on Cluster Computing and the Grid - CCGrid'03*. IEEE Press, 2003.

[5] Henri Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001*, pages 430–437, 2001.

[6] C. Dumitrescu and I. Foster. Gangsim: a simulator for grid scheduling studies. In *Proc. of the 5th IEEE Intl*

Symp. on Cluster Computing and the Grid - CCGrid'05, pages 1151–1158. IEEE Press, 2005.

[7] GSSIM. Grid scheduling simulator website. Web page available at http://www.gssim.org, last accessed in January 2012, 2012.

[8] W. M. Jones, L. W. Pang, D. Stanzione, and W. B. Ligon III. Characterization of bandwidth-aware metaschedulers for co-allocating jobs across multiple clusters. *Journal of Supercomputing*, 34:135–163, 2005.

[9] William M. Jones, John T. Daly, , and Nathan DeBardeleben. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 276–279. ACM, 2010.

[10] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz. Grid scheduling simulations with gssim. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems - Volume 02*, pages 1–8, 2007.

[11] Uri Lublin and Dror G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comput.*, pages 1105–1122, 2003.

[12] P. Yabo. Grid matrix website. available at http://research.nektra.com/Grid_Matrix, last accessed February, 2012, 2011.