Handling Very Large Platforms with the New SimGrid Platform Description Formalism

Marc-Eduard Frincu, Martin Quinson, Frédéric Suter

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Handling Very Large Platforms with the New SIMGrid *Platform Description Formalism*

Marc-Eduard Frincu — Martin Quinson — Frédéric Suter

N° 0348

2008

Thème NUM

ISSN 0249-0803 ISRN INRIA/RT--0348--FR+ENG

Handling Very Large Platforms with the New SimGrid Platform Description Formalism

Marc-Eduard Frincu, Martin Quinson, Frédéric Suter

Thème NUM — Systèmes numériques
Projet AlGorille

Rapport technique n° 0348 — 2008 — 24 pages

Abstract: Simulation of parallel and distributed applications is a very active research field as simulation allows for repeatable results, makes it possible to explore various platform scenarios at will, is not as labor-intensive or as costly as running experiments on a real platform, and often makes it possible to run enormous numbers of experiments quickly. If many simulation toolkits exist, most of them face the issue of the description of the large scale platforms that are currently deployed.

In this technical report we focus on the platform description format used by the SimGrid toolkit and present how we modify its DTD to tackle the scaling issues induced by large platforms. Experiments show a reduction by a 6,600 factor of the size of the XML file for a multi-cluster platform comprising 1,300 hosts. We also extend the existing DTD in order to integrate the description of other features like attaching arbitrary properties to resources or introducing randomness in the platform description and express the dynamic nature of a platform directly in the XML file.

Key-words: Simulation, Platform description, XML, SimGrid.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Gestion de très grandes plates-formes grâce au nouveau formalisme de description de SimGrid

Résumé : La simulation d'applications parallèles et distribuées est un domaine de recherche très actif puisque cela permet l'obtention de résultats reproductibles, rend possible l'exploration de multiples scénarios de plates-formes, n'est pas aussi coûteux en temps ou en ressources que de conduire des expériences sur une plate-forme réelle, et permet souvent l'exécution rapide d'un grand nombre d'expériences. Si plusieurs environnements de simulation existent, la plupart d'entre eux est confrontée au problème de la description des plates-formes à grande échelle actuellement déployées.

Dans ce rapport technique, nous nous intéressons au format de description de plates-formes utilisé par l'environnement SimGrid et présentons comment nous avons modifié sa DTD afin de résoudre le problème de gestion des plates-formes à large échelle. Les expériences montrent une réduction par un facteur 6 600 de la taille du fichier XML décrivant une plate-forme multi-grappe comprenant 1 300 hôtes. Nous avons également étendu la DTD existante afin d'intégrer la description de fonctionnalités supplémentaires telles que l'ajout de propriétés pour certaines ressources, l'introduction de caractères aléatoires dans la description ou l'expression de la nature dynamique de la plate-forme directement dans le fichier XML.

Mots-clés : Simulation, description de plates-formes, XML, SimGrid.

1 Introduction

One key challenge for mastering the large scale, heterogeneous and distributed platforms such as computing grids or Peer-to-Peer systems is the development of techniques for maximizing application performance of these multi-cluster platforms, by redesigning the applications and/or by using novel algorithms that can account for the composite and heterogeneous nature of the platform. Most research works in this area are based on simulation, as it allows for repeatable results, makes it possible to explore various platform scenarios at will, is not as labor-intensive or as costly as running experiments on a real platform, and often makes it possible to run enormous numbers of experiments quickly. Several toolkits exist for running these simulations, such as GridSim [2], OptorSim [1] or SimGrid [5]. Each of these tools has its own way to describe the platform configurations used by the simulation kernel. The different formats have in common the description of a platform as a set of computation and communication resources and a description of the network interconnection. OptorSim and SimGrid use external text files for such a description while GridSim defines the platform directly in the code of the simulator through the use of Java classes.

The most prominent characteristic of currently deployed computing platforms, be they Peer-to-Peer systems or computing grids, is their large scale. This characteristic raises a scaling issue of the platform description files as describing a platform comprising thousands of computing elements often lead to large text files (several MB) and hence a long parsing time.

In this technical report we focus on the platform description format used by the SimGrid toolkit and present how we modify its DTD to tackle the scaling issues induced by large platforms. We also extend the existing DTD in order to integrate the description of other features like attaching arbitrary properties to resources or introducing randomness in the platform description and express the dynamic nature of a platform directly in the XML file.

The remaining of this report is organized as follow. In Section 2 we describe how GridSim, OptorSim and SimGrid represent platform configurations for driving simulations. In Section 3 we detail the DTD of the 3.2 release and its limitations. In Section 4 we present the new features and the different improvements provided in the DTD of the 3.3 release. In Section 5 we evaluate the improvements allowed by the new DTD in terms of file size and parsing time. In Section 6 we present a few examples of the application of this work to model currently deployed grid platforms. Finally we give a summary of our propositions and some future work in Section 7.

2 Related Work on Platform Description

2.1 GridSim

In the GridSim[1] toolkit ([2]) the simulated platform configuration is described directly in the source code of the simulator. GridSim provides a Java API to declare the different kinds of resources and the network interconnections. The following example (extracted from `http://www.gridbus.org/gridsim/example/index.html`) illustrates the description of a platform comprising two machines.

Computing resources are represented by *Processing Elements (PEs)* which stand for CPUs running at a certain speed expressed in Million Instructions Per Second (MIPS) or in SPEC-like ratings. Such processing elements can be grouped to create *machines*. In the example (lines 6 and 11) two machines are created, each being made of two processing elements. GridSim allows users to specify different feature for a given machine such as the system architecture, the operation system, the timezone, the sharing policy or the load of the machine during different periods. Most of these options are used by the budget-centric simulators written in GridSim. Machines can then be grouped to form a Grid Resource (GR).

Creating resources in GridSim

```
1 MachineList mList = new MachineList();
2
3 PEList peList1 = new PEList();
4 peList1.add( new PE(0, 377) );
5 peList1.add( new PE(1, 377) );
6 mList.add( new Machine(0, peList1) );   // First Machine
7
8 PEList peList2 = new PEList();
9 peList2.add( new PE(0, 377) );
```

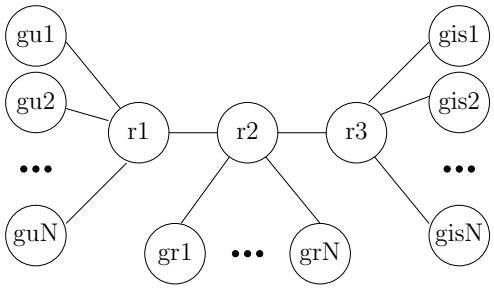
[1] `http://sourceforge.net/projects/gridsim/`

```
peList2.add( new PE(1, 377) );
mList.add( new Machine(1, peList2) );   // Second Machine

String arch = "Sun Ultra";      // system architecture
String os = "Solaris";          // operating system
double time_zone = 9.0;         // time zone this resource located
double cost = 3.0;              // the cost of using this resource

ResourceCharacteristics resConfig = new ResourceCharacteristics(
                arch, os, mList, ResourceCharacteristics.TIME_SHARED,
                time_zone, cost);

double baud_rate = 100.0;           // communication speed
long seed = 11L * 13 * 17 * 19 * 23 + 1;
double peakLoad = 0.0;       // the resource load during peak hour
double offPeakLoad = 0.0;    // the resource load during off-peak hr
double holidayLoad = 0.0;    // the resource load during holiday

...

gridRes = new GridResource(name, baud_rate, seed,
                resConfig, peakLoad, offPeakLoad, holidayLoad, Weekends,
                Holidays);
```

Once several Grid Resources are created as presented in the first example, it is possible to declare a network topopoly between them. A given GridSim topology can connect various entities such as GRs, but also Grid Users (GUs) and Grid Information Services (GISs).

Communication resources in GridSim include network links and routers. A network link is characterized by its bandwidth (in bits per second), its propagation delay (in milliseconds) and its maximum transmission unit (MTU, expressed in bytes). A router is associated to an advertising protocol such as RIP.

The following example shows a simple topology connecting several GUs, GISs and GRs as depicted in the following figure.



Defining a simple network topology between various resources

```
double baud_rate = 1e8;  // 100 Mbps
double propDelay = 10;   // propagation delay in millisecond
int mtu = 1500;          // max. transmission unit in byte

// create the routers
Router r1 = new RIPRouter("router1");   // router 1
Router r2 = new RIPRouter("router2");   // router 2
Router r3 = new RIPRouter("router3");   // router 3

// connect all user entities with r1 router
// For each host, specify which PacketScheduler entity to use.
NetUserGIS obj;
for (int i = 0; i < userList.size(); i++) {
  FIFOScheduler userSched = new FIFOScheduler("NetUserSched_"+i);
  obj = (NetUserGIS) userList.get(i);
```

```
16   r1.attachHost(obj, userSched);
17 }
18
19 // connect all resource entities with r2 router
20 // For each host, specify which PacketScheduler entity to use.
21 GridResource resObj;
22 for (int i = 0; i < resList.size(); i++) {
23   FIFOScheduler resSched = new FIFOScheduler("GridResSched_"+i);
24   resObj = (GridResource) resList.get(i);
25   r2.attachHost(resObj, resSched);
26 }
27
28 // then connect r1 to r2
29 // For each host, specify which PacketScheduler entity to use.
30 Link link = new SimpleLink("r1_r2_link", baud_rate, propDelay, mtu);
31 FIFOScheduler r1Sched = new FIFOScheduler("r1_Sched");
32 FIFOScheduler r2Sched = new FIFOScheduler("r2_Sched");
33
34 // attach r2 to r1
35 r1.attachRouter(r2, link, r1Sched, r2Sched);
36
37 // attach r3 to r2
38 FIFOScheduler r3Sched = new FIFOScheduler("r3_Sched");
39 link = new SimpleLink("r2_r3_link", baud_rate, propDelay, mtu);
40 r2.attachRouter(r3, link, r2Sched, r3Sched);
41
42 // attach regional GIS entities to r3 router
43 RegionalGIS gis;
44 for (int i = 0; i < gisList.size(); i++) {
45   FIFOScheduler gisSched = new FIFOScheduler("gis_Sched" + i);
46   gis = (RegionalGIS) gisList.get(i);
47   r3.attachHost(gis, gisSched);
48 }
```

More details on the description of resources with GridSim can be found at `http://www.gridbus.org/gridsim/example/index.html`.

This approach presents several drawbacks. First, the Java language is not well adapted to platform description, and a lot of syntaxic sugar is needed to describe even a simple platform. Then, since the description is in the source code directly, it makes it difficult to run the same code simulating a given algorithm on several platforms. It is still possible to do so, but it leaves an extra burden on the user's shoulder since he/she has to either write a sort of generator in his/her code, or write a parser for a specifically designed platform description formalism. Finally, it makes it almost impossible to import platform descriptions generated by well known generators such as BRITE [8] or GridG [7].

2.2 OptorSim

With OptorSim[2] [1], users can describe their simulated platforms in an external configuration file. An example of such a file (extracted from the OptorSim distribution) follows. Each line of this file describes a particular site, or Computing Element (CE), of the platform. Informations are given as several raw integers: The first ones are respectively the amount of CPUs, the processing power of each CPU (expressed in kSpec-Int2000), the number of Storage Elements (SEs) and their size (expressed in MB) of the considered site. The end of each line (starting on the fifth column) indicates what is the maximum bandwidth (in Mb/s) between this CE and each other CE in the platform. When considering all the lines in the file, we thus obtain a matrix representing the network topology. OptorSim also allows to use dynamic workload traces stored in external files using a comparable formalism.

[2] `http://sourceforge.net/projects/optorsim`

In the following example, eight sites do not present any computing elements. It can be easily explained by the fact that OptorSim was designed in the European project DataGrid with a strong focus on data storage and not on high performance computing.

Example of OptorSim platform description file

```
# A simple network configuration based on 10 sites, two of which have CEs.
# no of CEs, no of SEs, SE sizes, site vs site bandwidth
#
 0 1 10000 0. 0. 1000. 0. 0. 0. 0. 0. 0.
 0 1 10000 0. 0. 1000. 0. 0. 0. 0. 0. 0.
 1 1 10000 1000. 1000. 0. 1000. 1000. 0. 0. 0. 0. 0.
 0 1 10000 0. 0. 1000. 0. 0. 0. 1000. 0. 0. 0.
 0 1 10000 0. 0. 1000. 0. 0. 0. 0. 0. 0. 0.
 0 1 10000 0. 0. 0. 0. 0. 0. 0. 1000. 0. 0.
 0 1 10000 0. 0. 0. 1000. 0. 0. 0. 1000. 0. 0.
 1 1 10000 0. 0. 0. 0. 0. 1000. 1000. 0. 1000. 1000.
 0 1 10000 0. 0. 0. 0. 0. 0. 0. 1000. 0. 0.
 0 1 10000 0. 0. 0. 0. 0. 0. 0. 1000. 0. 0.
```

Compared to GridSim, the separation of the platform description into an external file ease the testing of the same algorithm on a set of different platforms. Unfortunately, the chosen formalism may be easy to parse for the machine, but it is very difficult to read and write for human beings, making the task of editing these files very error prone. Finally, the only topologies that this syntax allows to express are very simplistic: intra-sites are assumed to to have an infinite bandwidth while inter-sites communications always follow the shortest path in number of hops, even if these assumptions do not hold in real settings. The OptorSim formalism makes it impossible to express more realistic platforms.

2.3 SimGrid

The SimGrid project is almost 10 years old, and the platform description formalism naturally changed over the years. SimGrid v1 (released in 1999 – [3]), provided an API allowing to build the platform description in a manner very similar to what is done in GridSim. SimGrid v2 (released in 2003 – [4]) introduced the ability to express the platforms in external files, but the used formalism was not very user friendly (even if it were not as terse as what can be found in OptorSim). SimGrid v3 (released in 2005 – [5]) introduced an XML-based formalism for platform description.

The first versions of SimGrid intended to simulate scheduling heuristics on heterogeneous platforms such as networks of workstations. Although most of the users relied on relative small scale and simple topologies such as cliques, stars or buses, it was already possible to describe complex platforms presenting non-trivial routing schema (such as non-symmetric paths) and dynamic load traces.

The SimGrid version 3.3 described in this technical report differentiates itself from GridSim and OptorSim as it allows users to define their own routes explicitly and to attach various resources to machines such as memory, disk capacity, operating system, network card type, number of cpu cores etc. Also some degree of randomness can be added to the computing power of the individual hosts inside clusters. Another difference is in the way it represents resources. SimGrid platform files use the XML language in order to provide a clear semantics and intuitive names for the them. The defined semantics offers support for defining both single machines and clusters, for connecting them using fixed routes and for linking traces either extracted from real machines or user defined ones.

Another difference with GridSim is the use of external deployment files in both OptorSim and SimGrid to specify how processes are deployed over the platform. As for the description of the platform itself, OptorSim uses a terse and error-prone formalism for this while SimGrid uses an XML-based formalism. It is however out of the scope of this paper to describe this DTD, and we restrict ourselves to the platform description here.

Before describing in the next sections the respective DTDs of the 3.2 and 3.3 releases, we have to mention that part of the changes concern the renaming of some tags and attributes. For instance, the `<platform_description>` and `<network_link>` tag have been renamed as `<platform>` and `<link>` to reduce their verbosity. Other were misnamed like the `<cpu>` tag that becomes `<host>`. Finally a syntax more commonly used in XML documents has been adopted. The `name` attribute used by most the tags is now `id` and we rely on the `:ctn` variant of the `<link>` tag to replace the `<route_element>` tag. These new notations will be used in the remaining of this

document, even in the description of the 3.2 release DTD. It has to be noticed that conversion scripts have been written to ensure that SimGrid users can still perform simulations relying on existing platform configurations.

The units used in SimGrid are the following: the computing power of a host is expressed in number of floating operations per second (flop/s) while the bandwidth and the latency of a network link are respectively measured in bytes per second and seconds.

3 SimGrid 3.2 DTD

In this section we present the syntax and semantic of the different tags declared in the SimGrid DTD as they were available in the 3.2 release.

3.1 Specifying hosts

This presentation will start with the description of a minimal platform comprising of a single host named BOB whose CPU that can compute 500,000,000 flop/s (or 500 MFlop/s).

Specifying a host

```
1 <?xml version='1.0'?>
2 <!DOCTYPE platform SYSTEM "simgrid.dtd">
3 <platform version="1">
4   <host id="BOB" power="500000000"/>
5 </platform>
```

Before adding more hosts to this simplistic platform, we propose to see what are the optional attributes of the `<host>` tag (for sake of readability, the lines 1-2 of previous example are omitted in the subsequent ones).

Expressing dynamicity. It is also possible to seamlessly declare a host whose availability changes over time using the *availability_file* attribute and a separate text file whose syntax is exemplified below.

Adding a trace file

```
1 <platform version="1">
2   <host id="BOB" power="500000000"
3         availability_file="bob.trace" />
4 </platform>
```

Example of "bob.trace" file

```
1 PERIODICITY 1.0
2 0.0 1.0
3 11.0 0.5
4 20.0 0.8
```

At time 0, our host will deliver 500 Mflop/s. At time 11.0, it will deliver half, that is 250 Mflop/s until time 20.0 where it will will start delivering 80% of its power, that is 400 Mflop/s. Last, at time 21.0 (20.0 plus the periodicity 1.0), we loop back to the beginning and the host will deliver again 500 Mflop/s.

Changing initial state. It is also possible to specify whether the host is up or down by setting the *state* attribute to either `ON` (default value) or `OFF`.

Expliciting the default value "ON"

```
1 <platform version="1">
2    <host id="BOB"
3          power="500000000"
4          state="ON" />
5 </platform>
```

Host switched off

```
1 <platform version="1">
2    <host id="BOB"
3          power="500000000"
4          state="OFF" />
5 </platform>
```

Expressing churn. To express the fact that a host can change state over time (as in P2P systems, for instance), it is possible to use a file describing the time at which the host is turned on or off. An example of the content of such a file is presented on the right hand-side.

Adding a state file

```
1 <platform version="1">
2   <host id="BOB" power="500000000"
3         state_file="bob.fail" />
4 </platform>
```

Example of "bob.fail" file

```
1 PERIODICITY 10.0
2 1.0 -1.0
3 2.0 1.0
```

A negative value means *down* while a positive one means *up and running*. From time 0.0 to time 1.0, the host is on. At time 1.0, it is turned off and at time 2.0, it is turned on again until time 12 (2.0 plus the periodicity 10.0). It will be turned on again at time 13.0 until time 23.0, and so on.

3.2 Specifying inter-hosts network connections

To create a more complex platform with several hosts, we have to specify the network connection between the different hosts using the `<link>`, `<route>` and `<link:ctn>` tags.

3.2.1 Declaring network links

Network links represent one-hop network connections. They are characterized by their *id* and their *bandwidth*. The *latency* is optional with a default value of 0.0. For instance, we can declare a network link named LINK1 having bandwidth of 1Gb/s and a latency of $50\mu s$.

Example link

```
<link id="LINK1" bandwidth="125000000" latency="5E-5"/>
```

Expressing sharing policy. By default a network link is `SHARED`, that is if more than one flow go through a link, each get an equal share of the available bandwidth. Conversely if a link is defined as a `FATPIPE`, each flow going through this link will get all the available bandwidth, whatever the number of flows. The `FATPIPE` behavior allows to describe switches or Internet backbones.

```
<link id="SWITCH" bandwidth="125000000" latency="5E-5"
      sharing_policy="FATPIPE" />
```

Expressing dynamicity and failures. As for hosts, it is possible to declare links whose state, bandwidth or latency change over the time. In this case, the `bandwidth` and `latency` attributes are respectively replaced by the `bandwidth_file` and `latency_file` attributes and the corresponding text files.

```
<link id="LINK1"                 state_file="link1.fail"
      bandwidth="80000000"       latency=".0001"
      bandwidth_file="link1.bw" latency_file="link1.lat" />
```

It has to be noted that even if the syntax is the same, the semantic of bandwidth and latency trace files differs from that of host availability files. Those files do not express availability as a fraction of the available capacity but directly in bytes per seconds for the bandwidth and in seconds for the latency. This is because most tools allowing to capture traces on real platforms (such as NWS – [11]) express their results this way.

Example of "link1.bw" file

```
PERIODICITY 12.0
4.0 40000000
8.0 60000000
```

Example of "link1.lat" file

```
PERIODICITY 5.0
1.0 0.001
2.0 0.01
3.0 0.001
```

In this example, the bandwidth varies with a period of 12 seconds while the latency varies with a period of 5 seconds. At the beginning of simulation, the link's bandwidth is of 80,000,000 B/s (i.e., 80 Mb/s). After four seconds, it drops at 40 Mb/s, and climbs back to 60 Mb/s after eight seconds. It keeps that way until second 12 (ie, until the end of the period), point at which it loops its behavior (seconds 12-16 will experience 80 Mb/s, 16-20 40 Mb/s and so on). In the same time, the latency values are $100\mu s$ (initial value) on the $[0, 1[$ time interval, $1ms$ on $[1, 2[$, $10ms$ on $[2, 3[$, $1ms$ on [3,5[(i.e., until the end of period). It then loops back, starting at $100\mu s$ for one second.

3.2.2 Declaring routes

Routes represent the network path between hosts. They correspond to the list of one-hop links that go from a source host to a destination host. A route is defined by the names of the source (*src* attribute) and destination (*dst* attribute) and comprises links. In the following example, we only have one link in the route. The order of the links on the path is not relevant with most models used in SIMGrid[3].

[3]Only the GTNetS pseudo-model described in Section 3.2.3 takes the order of links in a path into account.

Example platform with two hosts

```
<platform version="1">
  <host id="BOB" power="1000000000"/>
  <host id="ALICE" power="500000000"/>

  <link id="LINK1" bandwidth="125000000" latency="5E-5"/>

  <route src="BOB" dst="ALICE">
    <link:ctn id="LINK1"/>
  </route>

  <route src="ALICE" dst="BOB">
    <link:ctn id="LINK1"/>
  </route>
</platform>
```

Expressing multi-hop routes. Routes between hosts can be more complex. For instance if BOB and ALICE both have their own connection to a common switch, the route between them will comprise three distinct network links.

Example platform with multi-path routing

```
<platform version="1">
  <host id="BOB" power="1000000000"/>
  <host id="ALICE" power="500000000"/>

  <link id="LINK_BOB" bandwidth="125000000" latency="5E-5"/>
  <link id="LINK_ALICE" bandwidth="125000000" latency="5E-5"/>
  <link id="SWITCH" bandwidth="125000000" latency="5E-5"
        sharing_policy="FATPIPE" />

  <route src="BOB" dst="ALICE">
    <link:ctn id="LINK_BOB"/>
    <link:ctn id="SWITCH"/>
    <link:ctn id="LINK_ALICE"/>
  </route>

  <route src="ALICE" dst="BOB">
    <link:ctn id="LINK_ALICE"/>
    <link:ctn id="SWITCH"/>
    <link:ctn id="LINK_BOB"/>
  </route>
</platform>
```

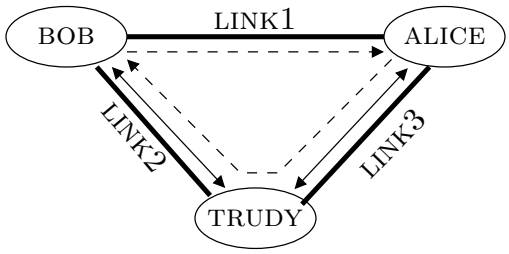
Expressing non-symmetric routes. In the previous example, the route between BOB and ALICE is symmetric, ie. the links used by the data flow between BOB and xALICE does not depend on the direction. It is however possible to describe non-symmetric routes as follows.

Example platform with non-symmetric routes

```
<platform version="1">
  <host id="BOB" power="1000000000"/>
  <host id="ALICE" power="500000000"/>
  <host id="TRUDY" power="250000000"/>
```

```
5
6   <link id="LINK1" bandwidth="125000000" latency="5E-5"/>
7   <link id="LINK2" bandwidth="125000000" latency="5E-5"/>
8   <link id="LINK3" bandwidth="125000000" latency="5E-5"/>
9
10  <route src="BOB" dst="ALICE">  <link:ctn id="LINK1"/></route>
11  <route src="ALICE" dst="BOB">
12    <link:ctn id="LINK2"/>
13    <link:ctn id="LINK3"/>
14  </route>
15
16  <route src="BOB" dst="TRUDY">  <link:ctn id="LINK2"/></route>
17  <route src="TRUDY" dst="BOB">  <link:ctn id="LINK2"/></route>
18
19  <route src="TRUDY" dst="ALICE"><link:ctn id="LINK3"/></route>
20  <route src="ALICE" dst="TRUDY"><link:ctn id="LINK3"/></route>
21 </platform>
```

This makes sure that data from ALICE to BOB goes through links LINK3 and LINK2 while data from BOB to ALICE goes directly through LINK1 (the non-symmetric path is dashed on the following schema).



3.2.3 Specifying routers

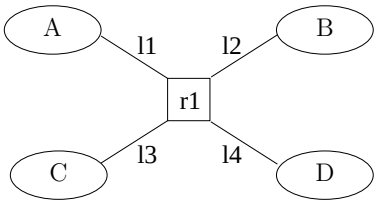
For sake of completeness, we now present the formalism allowing to express routers in SimGrid, even if it is only needed in very rare circumstances. Most of the SimGrid platform descriptions do not contain any routers and routes between hosts are modeled as sets of individual links as we saw.

However, expressing routers becomes mandatory when using the bindings to the GTNetS packet-level simulator instead of the native analytical network model implemented in SimGrid. Routers are naturally an important concept in GTNetS since the way they run the packet routing algorithms is actually simulated. Instead, the SimGrid's analytical models aggregate the routing time with the transfer time.

Rebuilding a graph representation only from the route information turns to be a very difficult task, because of the missing information about how routes intersect. That is why we introduced a `<router>` tag, which is simply used to express these intersection points. The only attribute accepted by this tag an id.

It is important to understand that the `<router>` tag is only used to provide topological information. In the following example, the paths AB and CD do not share any resource (because $\{l1, l2\} \cap \{l3, l4\} = \emptyset$) and thus do not impact on each other, unless when using the GTNetS bindings. More information on these bindings can be found in [6].

```
1 <route src="A" dst="B">
2   <link:ctn id="l1"/><router id="r1"/><link:ctn id="l2"/>
3 </route>
4
5 <route src="C" dst="D">
6   <link:ctn id="l3"/><router id="r1"/><link:ctn id="l4"/>
7 </route>
```



3.3 Implementation

The SimGrid XML parser is implemented using the **FlexML**[4] toolkit. As such it follows the SAX approach and relies on events during the parsing process. This approach presents an advantage over standard DOM-style parsers because it does not require to load the entire XML tree in the memory before actually parsing it. It thus leads to a lower memory consumption and faster parsing procedures. This becomes very important in the case of large XML files such as those describing large computing platforms.

The specificity of FlexML over other SAX-based parsing solutions is that it does not constitute a parsing library by itself. Instead, it converts the DTD file into a parser specification usable with the classical Flex parser generator. In turn, the generator changes this specification into a highly efficient C parser inducing no extra dependency at runtime. This is important in our context since part of SimGrid can be used as a grid middleware (see [9]), and dependencies have then to be avoided to ease the toolkit installation.

Because of scalability issues both in the old DTD and in previous implementations of FlexML, a mechanism was introduced to allow the users to completely bypass the FlexML parser by providing its own parsing function accepting a different formalism than the classical SimGrid one. The details of how to implement such a function are beyond the scope of this report, and can be found at `http://simgrid.gforge.inria.fr/doc/faq.html#faq_flexml_bypassing`. Quickly, it consists in filling specific variables and call directly the C functions representing the SAX callbacks.

3.4 Limitations

Verbosity. The main limitation of the 3.2 DTD release that first motivated this work is the high verbosity of the produced XML files. This comes from the need to describe every single route of the platform. For instance, in the case of a clique comprising 100 hosts, the description requires 9,900 route definitions (100×99). If there is almost nothing to improve for such a flat platform, it is not the same for more hierarchical platforms. Assume a platform comprising of 4 homogeneous clusters, each made of 25 hosts. Hosts in a cluster are interconnected through a single switch and the 4 clusters are themselves interconnected through a single backbone. In this case, we have the same number of hosts as in the flat platform and the same number of routes (if we assume a fully connected network). But there, each intra-cluster route includes 3 different links (source own link – switch – destination own link) and each inter-cluster route includes 5 different links (source own link – source switch – backbone – destination switch –destination own link) leading to bigger XML description. Moreover most of this information is redundant as for each pair of nodes located in two different clusters, the switch–backbone–switch part is common. We can also find redundant information inside a cluster, as we describe 25 different hosts with exactly the same characteristics that just differ by their ids.

Inefficient parsing. Despite the use of a fast and optimized parser generated by FlexML, SimGrid 3.2 does parse a platform file not once, but four times. This is because of the modular design of the simulation kernel, where the hosts and the network are modeled by different sub-modules. In v3.2, each sub-module parsed the whole file once looking for relevant informations. In addition, the network sub-modules' parsing implementation were not optimized at all. Before retrieving the routing information, one has to allocate the needed space for the routing table, but this can only be done when knowing the amount of nodes in the platform. Then, one also need to know each existing link before parsing the routing information. Since the order of declaration is not enforced by the DTD, a file can declare hosts after declaring routes. That is why the SimGrid v3.2 parser reads the file four times: one to initialize the host sub-model, and three for the network one (first pass to get the amount of nodes, second one to get link information and third one to get route information).

[4] `http://flexml.sourceforge.net`

4 SimGrid 3.3 DTD

In this section we detail the different modifications we made to the SimGrid DTD. The first change was to solve the main limitation of previous version (expressed in Section 3.4) by compacting the platform description (as described in Section 4.1). The second one, described in Section 4.2, allows to declare the resource availability directly in the platform file. This section also details two new features. The first one consists in offering the possibility to attach arbitrary properties to any kind of resources (described in Section 4.3). The second one, described in Section 4.4, allows users to define random generators and use them to set the values of some attribute (`power` for a host or `bandwidth` for a link). Finally, Section 4.5 briefly explains how we optimized the parser by parsing the files only once.

To distinguish files following the 3.3 DTD from those following the 3.2 DTD, we change the value of the `version` attribute of the `<platform>` tag. Consequently all the examples presented in this section will begin by `<platform version="2">`.

4.1 Compacting the Platform Description

Name sets. Among the limitations mentioned in section 3.4 was that when declaring several hosts belonging to a single homogeneous cluster, everything but the name of each host is common. We thus introduce the new `<set>` tag which allows users to define a lists of names. All the names in this list share a common `prefix` and a common `suffix`. The number of items of the list and the part of the name that distinguishes the different names are defined through the regular expression in the `radical` attribute. In the following example we define a name set including 5 items (`bob-0.hamburger.edu`, `bob-2.hamburger.edu`, `bob-3.hamburger.edu`, `bob-4.hamburger.edu`, and `bob-5.hamburger.edu`).

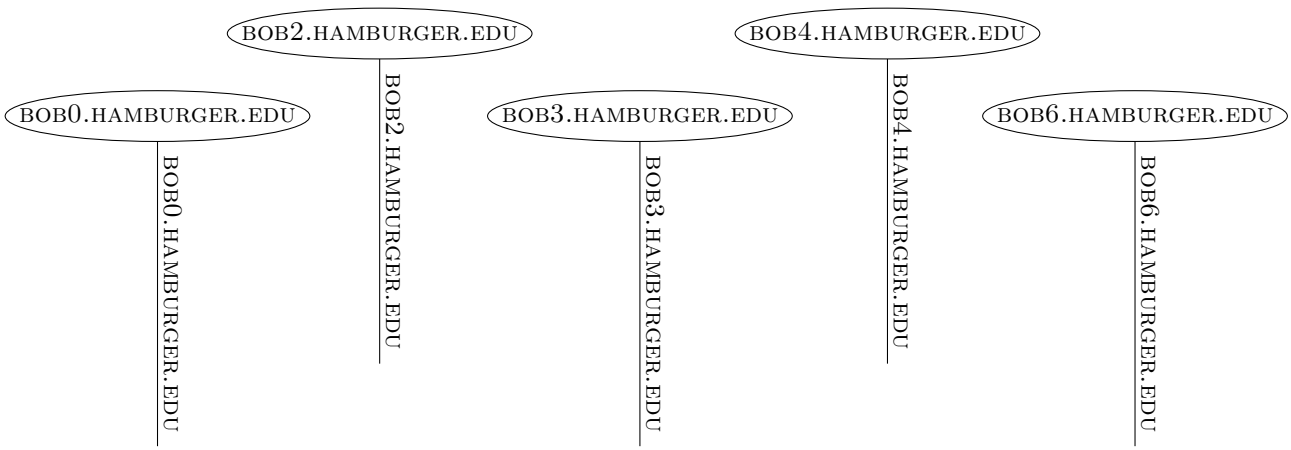
Defining a name set

```
1 <set id="myCluster" prefix="bob" suffix=".hamburger.edu" radical="0,2-4,6"/>
```

Foreach loops. Using these name sets, it is easy to declare a homogeneous set of hosts with the newly introduced `<foreach>` tag. The meaning of this tag, whose sole attribute is a name set (`set_id`), is close to that of a classic for loop. For each item in the name set, the different tags declared between the `<foreach>` and `</foreach>` tags will be instantiated using one of the names of the set as id. In the following example, we declare a set of 5 homogeneous hosts (running at 1Gflop/s) and five homogeneous network links (with a 1Gb/s bandwidth and a $50\mu s$ latency).

Defining hosts and links using sets and foreach

```
1 <platform version="2">
2   <set id="myCluster" prefix="bob" suffix=".hamburger.edu" radical="0,2-4,6"/>
3   <foreach set_id="myCluster">
4     <host id="$1" power="1000000000"/>
5     <link id="$1" bandwidth="125000000" latency="5E-5"/>
6   </foreach>
7 </platform>
```



In this example we rely on an additional special symbol, **$1** to refer to the id of the parent tag. The XML description corresponding to this example using the DTD of the 3.2 release is as follow.

Corresponding example in the 3.2 DTD

```
1  <platform version="1">
2    <host id="bob0.hamburger.edu" power="1000000000"/>
3    <host id="bob2.hamburger.edu" power="1000000000"/>
4    <host id="bob3.hamburger.edu" power="1000000000"/>
5    <host id="bob4.hamburger.edu" power="1000000000"/>
6    <host id="bob6.hamburger.edu" power="1000000000"/>
7
8    <link id="bob0.hamburger.edu" bandwidth="125000000" latency="5E-5"/>
9    <link id="bob2.hamburger.edu" bandwidth="125000000" latency="5E-5"/>
10   <link id="bob3.hamburger.edu" bandwidth="125000000" latency="5E-5"/>
11   <link id="bob4.hamburger.edu" bandwidth="125000000" latency="5E-5"/>
12   <link id="bob6.hamburger.edu" bandwidth="125000000" latency="5E-5"/>
13 </platform>
```

Declaring complete clusters. In case of such a homogeneous cluster comprising n hosts, it is also possible to compact the declaration of the intra-cluster routes. As explained before, if the n hosts are interconnected through a single, the declaration of $n \times (n-1)$ routes made of three links is needed. In the 3.3 DTD we introduce the new `<route:multi>` tag allowing users to declare a common route for the members of a name set. In the following example we add a switch to the previous set of hosts and declare all the routes connecting each pair of hosts thanks to one single `<route:multi>` tag.

Defining intra-cluster routes

```
1  <platform version="2">
2    <set id="myCluster" prefix="bob" suffix=".hamburger.edu" radical="0,2-4,6"/>
3
4    <foreach set_id="myCluster">
5      <host id="$1" power="1000000000"/>
6      <link id="$1" bandwidth="125000000" latency="5E-5"/>
7    </foreach>
8
9    <link id="bob_backbone" bandwidth="2250000000" latency="5E-4"
10         sharing_policy="FATPIPE"/>
11
12   <route:multi src="myCluster" dst="myCluster">
13     <link:ctn id="$src"/>
14     <link:ctn id="bob_backbone"/>
15     <link:ctn id="$dst"/>
16   </route:multi>
17 </platform>
```

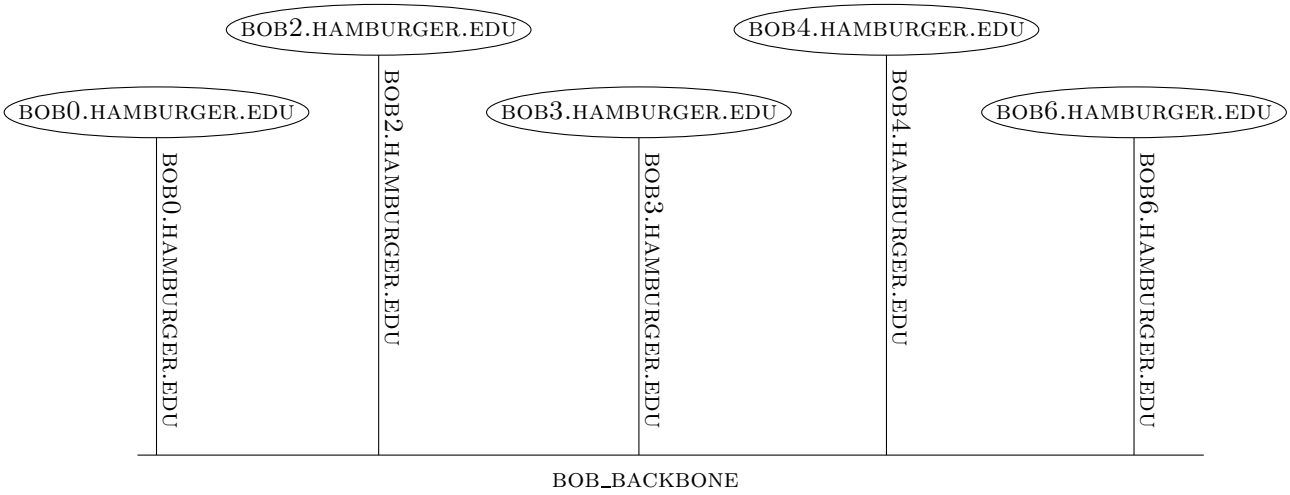
The semantic of the `<route:multi>` tag is that for each pair of elements in the name sets $src \times dst$, the links specified within the tag must be **added** to the interconnection route. In this particular example, both name sets are myCluster since we want to specify the routing from the cluster to itself.

The special symbol **$src** (resp. **$dst**) is replaced by the name of the host from the source set (resp. destination set). Our example thus rely on the fact that the `<foreach>` tag on lines 4-7 gave the same name to both the host and the link in each iteration.

The cluster tag. Such a declaration actually corresponds to the definition of a homogeneous cluster. Since this construction happens very often in real platforms, we introduce the `<cluster>` tag. It defines a cluster of n processors, each of them being connected to a common backbone by a private link. The backbone in turn is connected to the outer world. The name of the backbone is obtained by appending "_bb" to the cluster name (ie, "myCluster_bb" here). The private links are named after the host they serve as previously. The previous example can thus be written in a more concise way as follow.

The cluster tag

```
<cluster id="myCluster" prefix="bob" suffix=".hamburger.edu"
         radical="0,2-4,6" power="1000000000" bw="125000000" lat="5E-5"
         bb_bw="250000000" bb_lat="5E-4"/>
```

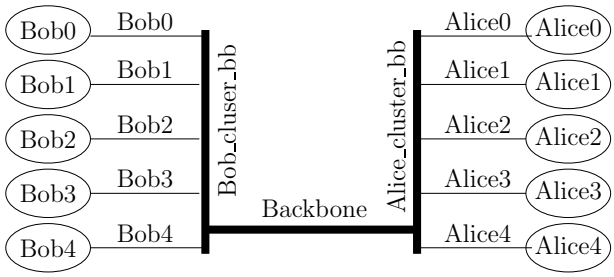


With this tag we defined a homogeneous cluster of 5 hosts whose names are bobX.hamburger.edu, with $X \in \{0, 2, 3, 4, 6\}$. Each host runs at 1Gflop/s and is connected to a private link having a bandwidth of 1Gb/s and a latency of $50\mu s$ (given by the **bw** and **lat** attributes). Each pair of hosts is connected through their private links and an internal backbone whose bandwidth is 2Gb/s and latency is $500\mu s$ (given by the **bb_bw** and **bb_lat** attributes). By default this internal backbone link follows the `fatpipe` sharing policy.

Inter-cluster routing. When the platfom configuration comprises several clusters, it is very simple to declare all the routes between each pair of hosts. In the following example we describe a platform made of two clusters (bob_cluster and alice_cluster. Remember that the `<cluster>` hides the declaration of the inter-cluster routes and the beginning of the routes between a host within a cluster and the outer world. Consequently to connect one cluster to another, we just have to append the end of the routes (the inter-cluster backbone link and the internal routes of the destination cluster, given by `$dst`). This declaration has to be repeated for each pair of clusters.

Defining inter-cluster routes

```
<platform version="2">
  <cluster id="bob_cluster" prefix="bob" suffix=".hamburger.edu"
           radical="0-4" power="1000000000" bw="125000000" lat="5E-5"
           bb_bw="250000000" bb_lat="5E-4"/>

  <cluster id="alice_cluster" prefix="alice" suffix=".crepe.fr"
           radical="0-4" power="2000000000" bw="125000000" lat="5E-5"
           bb_bw="250000000" bb_lat="5E-4"/>

  <link id="backbone" bandwidth="1250000000" latency="5E-4"/>

  <route:multi src="bob_cluster" dst="alice_cluster">
    <link:ctn id="backbone"/>
    <link:ctn id="$dst"/>
  </route:multi>

  <route:multi src="alice_cluster" dst="bob_cluster">
    <link:ctn id="backbone"/>
    <link:ctn id="$dst"/>
  </route:multi>
</platform>
```



Symmetry of inter-cluster routes. In the previous example, each route from the BOB_CLUSTER to the ALICE_CLUSTER is symmetric. In such a case it is possible to save the declaration of the routes in one of the direction thanks to the **symmetric** attribute of the `<route:multi>` tag. It accepts two different values: either NO (default, meaning that the route is not symmetric) and YES (meaning that the route is symmetric and that each link contained in the tag must be added to both directions).

Defining symmetric inter-cluster routes

```
<platform version="2">
  <cluster id="BOB_CLUSTER" prefix="BOB" suffix=".HAMBURGER.EDU"
           radical="0-4" power="1000000000" bw="125000000" lat="5E-5"
           bb_bw="250000000" bb_lat="5E-4"/>

  <cluster id="ALICE_CLUSTER" prefix="ALICE" suffix=".CREPE.FR"
           radical="0-4" power="2000000000" bw="125000000" lat="5E-5"
           bb_bw="250000000" bb_lat="5E-4"/>

  <link id="BACKBONE" bandwidth="1250000000" latency="5E-4"/>

  <route:multi src="BOB_CLUSTER" dst="ALICE_CLUSTER" symmetric="yes">
    <link:ctn id="$src"/>
    <link:ctn id="BACKBONE"/>
    <link:ctn id="$dst"/>
  </route:multi>
</platform>
```

Routing between clusters and hosts. It is also possible to define the routes between a cluster and a single host using the same kind of declaration. In the following example each host of BOB_CLUSTER is connected to TRUDY.

Connecting a cluster and a host

```
<platform version="2">
  <cluster id="BOB_CLUSTER" prefix="BOB" suffix=".HAMBURGER.EDU"
           radical="0-4" power="1000000000" bw="125000000" lat="5E-5"
           bb_bw="250000000" bb_lat="5E-4"/>

  <host id="TRUDY" power="250000000"/>

  <link id="BACKBONE" bandwidth="1250000000" latency="5E-4"/>

  <route:multi src="BOB_CLUSTER" dst="TRUDY" symmetric="yes">
    <link:ctn id="BACKBONE"/>
    <link:ctn id="$dst"/>
  </route:multi>
</platform>
```

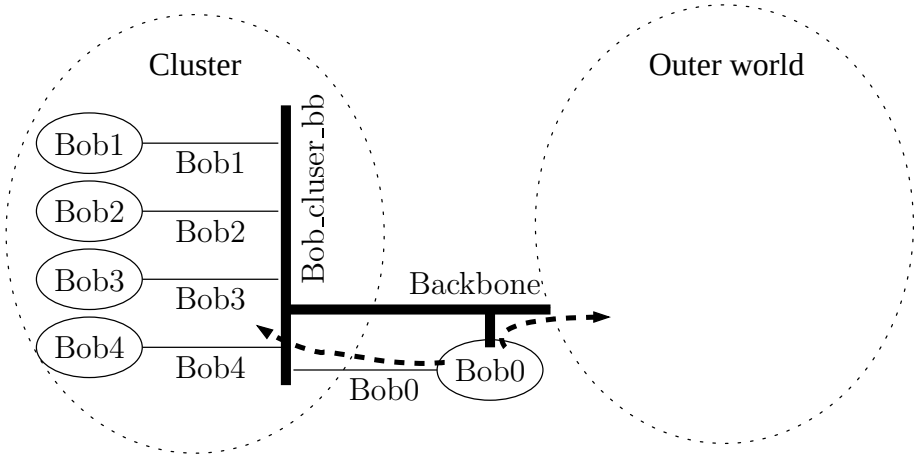
Overriding routes. It is possible to specify the connectivity of a distinguished node through the use of the `action` attribute of the `<route:multi>` tag. The default value of this attribute is `postpend` meaning that the definition of `<route:multi>` tag will *append* new links to an existing partial route. In order to modify an already

completed route the `action` attribute is set to `override`. In the following example, BOB0.HAMBURGER.EDU is first connected to every other host of the platform (including those of BOB_CLUSTER) through the inter-cluster backbone. The new special symbol `$*` represent all the resources of a kind declared in the XML file. So, the `<route:multi>` tag on lines 8-11 could be read as "redefine the route from BOB0 to every host (even those who have not been decelared yet) as using this path".

Then we have to redefine the intra-cluster routes from BOB0 to go through the intra-cluster backbone (which is automatically named BOB_CLUSTER_BB).

Overriding a route

```
1  <platform version="2">
2    <cluster id="BOB_CLUSTER" prefix="BOB" suffix=".HAMBURGER.EDU"
3             radical="0-4" power="1000000000" bw="125000000" lat="5E-5"
4             bb_bw="250000000" bb_lat="5E-4"/>
5
6    <link id="BACKBONE" bandwidth="1250000000" latency="5E-4"/>
7
8    <route:multi src="BOB0.HAMBURGER.EDU" dst="$*" action="override">
9      <link:ctn id="BACKBONE"/>
10     <link:ctn id="$dst"/>
11   </route:multi>
12
13   <route:multi src="BOB0.HAMBURGER.EDU" dst="BOB_CLUSTER" action="override">
14     <link:ctn id="$src"/>
15     <link:ctn id="BOB_CLUSTER_BB"/>
16     <link:ctn id="$dst"/>
17   </route:multi>
18 </platform>
```



4.2 Describe Availabilities in the XML File

Until version 3.2 availabilities where added strictly inside host and link tags using the `availability_file`, `state_file`, `bandwidth_file` or `latency_file` attributes and external text files, as described pages 7 and 8.

In the proposed 3.3 DTD, we extended this mechanism to allow the definition of traces directly inside the XML file, and the binding of a certain trace to its corresponding host or link at the end of the parsing. This can be used to define several trace files for the same platform, describing different time frames for example. Each trace file would then include the main platform file, declare all needed traces, and attach the traces to the elements. With previous formalism, users add to change the main platform description to achieve the same results.

Defining availabilities inside the XML file

```
1 <host id="BOB" power="1000000000"/>
2
3 <trace id="MYTRACE" periodicity="1.0">
4   0.0 1.0
```

```
  11.0 0.5
  20.0 0.8
</trace>

<trace:connect trace="myTrace" element="bob" kind="POWER"/>
```

The `<trace:connect>` tag connects the trace which id is given as *trace* attribute to the link or host which id is given as *element* attribute. The *kind* attribute describes what kind of trace is to be connected. Its value is one of: HOST_AVAIL, POWER, LINK_AVAIL, BANDWIDTH or LATENCY. Of course, when passing either HOST_AVAIL or POWER as *kind*, you should pass a `<host>` id as *element* while the other possible values of *kind* imply a `<link>` id as *element*.

It is also possible to declare traces in external files, and still connect them after element creation, thanks to the `<trace>` tag.

To ensure a backward compatibility with the 3.2 DTD, in which the traces were defined in external text files, we also offer the possibility to declare such traces (using the `file` attribute of the tag) and connect them to host or link resources. The external files rely on the same syntax and semantic as before.

Defining availabilities in an external file

```
<host id="bob" power="1000000000"/>

<trace id="myTrace" file="bob.trace">
<trace:connect element="bob" kind="power" trace="myTrace" />
```

4.3 Properties

There is many cases where users could need a way to add extra data to the platform elements. For example, one could add data about the disk and memory of each host for a matchmaking scheduling heuristic using this information, or the operation system name for studies on worm propagation in P2P systems.

Attaching properties to hosts and links. SimGrid 3.3 DTD introduce a new `<prop>` tag to do so. It can be used to build dictionaries of properties with the *key* and `data` attributes. It can be used to attach arbitrary data to any host or link of the platform. The user is free to declare any property to be associated to a resource, with the value he/she wants, but he/she is also responsible of their management. SimGrid only provides a way to add arbitrary data in the XML file and functions to retrieve them after parsing.

Attaching properties to a host

```
<platform version="2">
  <host id="bob" power="1000000000"/>
    <prop key="memory" value="1000000000"/>
    <prop key="disk" value="80E9"/>
    <prop key="OS" value="Linux 2.6.22-14"/>
  </host>
</platform>
```

In this example, 1GB of memory, 80GB of disk and a Linux system were attached to host BOB.

Properties and clusters. It is also possible to attach a property to the different items composing a name set, using the `<foreach>` tag. In the following example, we specify that each host of BOB_CLUSTER has a dual-core processor and runs a Linux system.

Attaching properties to a name set

```
<set id="bob_cluster" prefix="bob-" suffix=".hambuger.edu" radical="0-3"/>
<foreach set_id="bob_cluster">
  <host id="$1" power="1000000000">
    <prop id="cores" value="2"/>
    <prop id="os" value="Linux"/>
  </host>
  <link id="$1" bandwidth="1250000000" latency="5E-5"/>
</foreach>
```

Overriding values. It is also possible the change a property value for one or more resources later in the XML file using the same property method. For instance, if the first host of BOB_CLUSTER is actually the front-end of that cluster and is not a dual-core but a quad-core, the following tag allows to modify the property of that special host.

Overriding the properties of a name set item

```
<host id="BOB-0.HAMBURGER.EDU" power="1000000000">
  <prop id="cores" value="4"/>
</host>
```

Retrieving values. The informations stored in the XML file can be retrieved at run time as a dictionary of values (of type xbt_dict_t, which is the SIMGrid dictionary data container). The API to do so changes slightly with the chosen SIMGrid interface. Only SimDag allows to retrieve link informations while MSG and GRAS are limited to host informations. This is because MSG and GRAS do not identify individual links at all, but only network paths. Moreover, the such ability were also added to the deployment files in order to attach properties to processes. We thus also list here the API to retrieve properties of processes.

SimDag interface

```
xbt_dict_t SD_link_get_properties(SD_link_t link);
const char*SD_link_get_property_value(SD_link_t link, const char* name);

xbt_dict_t SD_workstation_get_properties(SD_workstation_t workstation);
const char*SD_workstation_get_property_value(SD_workstation_t workstation,
                                             const char* name);
```

MSG interface

```
xbt_dict_t MSG_host_get_properties(m_host_t host);
const char*MSG_host_get_property_value(m_host_t host, const char*name);

xbt_dict_t MSG_process_get_properties(m_process_t process);
const char*MSG_process_get_property_value(m_process_t process,const char*name);
```

GRAS interface

```
xbt_dict_t gras_process_properties(void);
const char*gras_process_property_value(const char* name);

xbt_dict_t gras_os_host_properties(void);
const char*gras_os_host_property_value(const char* name);
```

4.4 Random Generators

To allow users to define a single XML file describing a set of platforms with the same structural properties (*e.g.,* number of hosts or clusters or interconnection topology) but in which attribute values (computing power of hosts, network link bandwidth, ...) can vary between instances, we introduce a new `<random>` tag in the 3.3 DTD. This tag allows to create a random generator, whose name is given by the `id` attribute, that rely on a standard function (such as `drand48`) to generate a random value. The name of the external random generator is given in the `generator` attribute. It is possible to precise which number to select in the generated stream by setting the `seed` attribute to the appropriated value. Then as random generators often produce values in the [0..1[interval, we use the `min` and `max` attributes to translate the generated random number into the [`min`..`max`[interval. The values of these two attributes are expressed in the unit of the aimed resource feature (*i.e.,* flops for host power, bytes for network bandwidth and seconds for network latency). Finally the `mean` and `std_deviation` attributes enforce the generator to produce values within the [`mean` − `std_deviation`; `mean` + `std_deviation`] interval.

Declaring a random generator

```
<random id="MYRANDOMGENERATOR" generator="DRAND48" seed="0"
        min="1000000000" max="2000000000" mean="1600000000"
        std_deviation="12"/>
```

Once a random generator has been declared, it can be used to set the value of the corresponding attribute of a single resource or to each of the items in a name set as shown in the following example.

Using a random generator for host's power

```
<platform version="2">
  <random id="myRandomGenerator" generator="DRAND48" seed="0"
          min="1000000000" max="2000000000" mean="1600000000"
          std_deviation="12" />

  <set id="bobcluster" prefix="bob-" suffix=".hamburger.edu" radical="0-3"/>

  <host id="bob" power="$rand(myRandomGenerator)"/>

  <foreach set_id="bob_cluster">
    <host id="$1" power="$rand(myRandomGenerator)"/>
    <link id="$1" bandwidth="125000000" latency="5E-5"/>
  </foreach>
</platform>
```

The `$rand()` function is a reference to the extern function defined by the `generator` attribute of the `<random>` tag. When used within a `<foreach>` tag a random generator produces a stream of random values. Consequently each host of `bob_cluster` will have a different power. This constitutes a convenient way to declare an heterogeneous cluster.

4.5 Further Parsing Process Optimization

As mentioned in Section 3.4, the parser of the 3.2 release had to parse the platform description files in several passes. Enabling a one-pass parsing of the XML implies that an order on the declaration of the different resources has to be enforced and that a mechanism allowing to temporarily store the parsed data is needed. The creation of the SimGrid data structures will only happen after the completion of the file parsing. This however required to modify the parsing mechanism of the simulation kernel. The main change was to allow more than one function to be attached to each of the SAX events. For instance the `</platform>` closing tag will trigger several functions to create the routes according to data stored by the `<route>` and `<route:multi>` tags, bind traces to hosts and links, etc.

Allowing more than one function to be assigned to a tag event also means that more flexibility can be added and each implemented simulation model can define its own sets of functions.

5 Evaluation

In this section we aim at measuring the improvements coming from the newly defined DTD in terms of file size and parsing speed. To conduct our experiments we used two kinds of platforms : two flat platforms whose interconnection topology has been generated by the Tiers graph generator and the Grid'5000[5] hierarchical multi-cluster platform. We only consider the description of the different hosts, network link and routes without any additional property or dynamic feature.

Concerning the XML file size we distinguish two levels of improvement. One comes from the renaming of some tags and attributes detailed in Section 2.3 while the other comes from the use of the compact `<cluster>` and `<route:multi>` tags of the 3.3 DTD. Table 1 shows the results for flat and multi-cluster platform configurations.

Platform	Topology	# hosts	3.2 DTD	3.2 DTD + renaming	3.3 DTD
Tiers generated	Flat	181	10,100	7,800	7,800
Tiers generated	Flat	300	73,900	55,500	55,500
Grid 5000 subset	Multi-clusters	300	26,000	21,000	6.4
Grid 5000 subset	Multi-clusters	615	117,000	94,700	18.1
Grid 5000 whole	Multi-clusters	1,300	520,000	425,000	78.8

Table 1: Comparison of the XML file size (in KBytes).

[5] `http://www.grid5000.fr`

We can see that for the Tiers generated platforms, the reduction of the size of the corresponding XML file only comes from the renaming the some of the tags and attributes. These modifications allow to save more than 20% (22.8% and 25,6% respectively) on the file size. However using the new tags of the 3.3 DTD does not reduce the size further as no clustering of host or route aggregation is possible in these flat platforms. For the platforms derived from Grid'5000, the renaming also allows a size reduction of about 20% (19.2%, 19.1% and 18.3% respectively) but as Grid'5000 is a multi-cluster platform, the size of the XML files is reduced by an order of magnitude thanks to the `<cluster>` and `<route:multi>` tags. For the description of the whole platform, the file is 6,600 times smaller written in the 3.3 DTD than it was using the 3.2 DTD. Furthermore it has to be noticed that these files are used as examples for the DTD formalism. We thus privileged their readability over there compression. That is why these files do not take advantage of the `symmetric` attribute and of the `$*` attribute value. A further reduction should be possible but would hinder file modification by users.

Concerning the parsing time reduction we distinguish four different descriptions of our test platforms : the original 3.2 DTD; the same DTD with renaming and with the implementation of the one-pass parsing; the 3.3 DTD without the `<cluster>` tag; and the same DTD with the `<cluster>` tag. For the two descriptions using the 3.3 DTD, the one-pass parsing is applied.

Platform	# hosts	3.2 DTD	3.2 DTD renaming + one pass	3.3 DTD with `<route:multi>` w/o `<cluster>`	3.3 DTD w/ `<cluster>`
Tiers generated	181	failed	1.5	1.1	1.1
Tiers generated	300	failed	7.9	4.8	4.8
Grid 5000 subset	300	failed	4.8	2.4	1.8
Grid 5000 subset	615	failed	8.4	6.4	5.2
Grid 5000 whole	1,300	failed	1,410	40	31

Table 2: Comparison of the XML file parsing time (in seconds).

The main constatation is that all the files failed to be parsed without the one-pass parsing. For all platforms but the whole Grid'5000 description, the one-pass parsing allows reasonable parsing times that are improve by the use of the `<route:multi>` and `<cluster>` tags. The main improvement coming from the compacting of the routing description is for large scale platforms (more than a thousand hosts). For the whole Grid'5000 description the parsing time is 45 times faster using the newly introduced tags than with the previous DTD.

6 Applications

Thanks to the new platform description formalism introduced in this technical report we started to build a repository of real world platform configurations. For instance we described two experimental grid platforms : DAS3[6] from Netherlands and Grid'5000 from France[7]. Figure 1 shows a representation of these two platforms generated from the corresponding XML files.

We also described three productions platforms, shown in Figure 2, used for the exploitation of the data produced by the Large Hadron Collider (LCG). The network topology and information for LCG, EGEE and GridPP platforms were taken from [10].

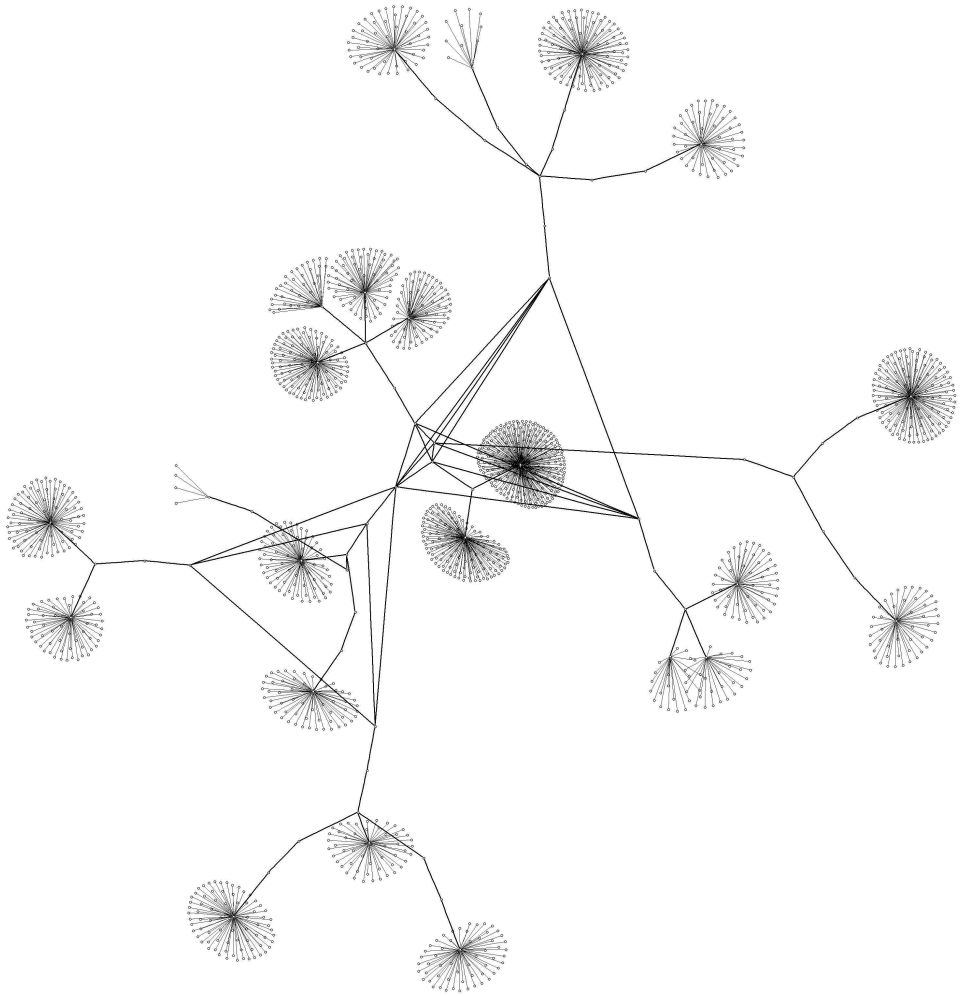
All these platform files can be found at `http://simgrid.gforge.inria.fr/doc/platforms.html`.

7 Conclusion

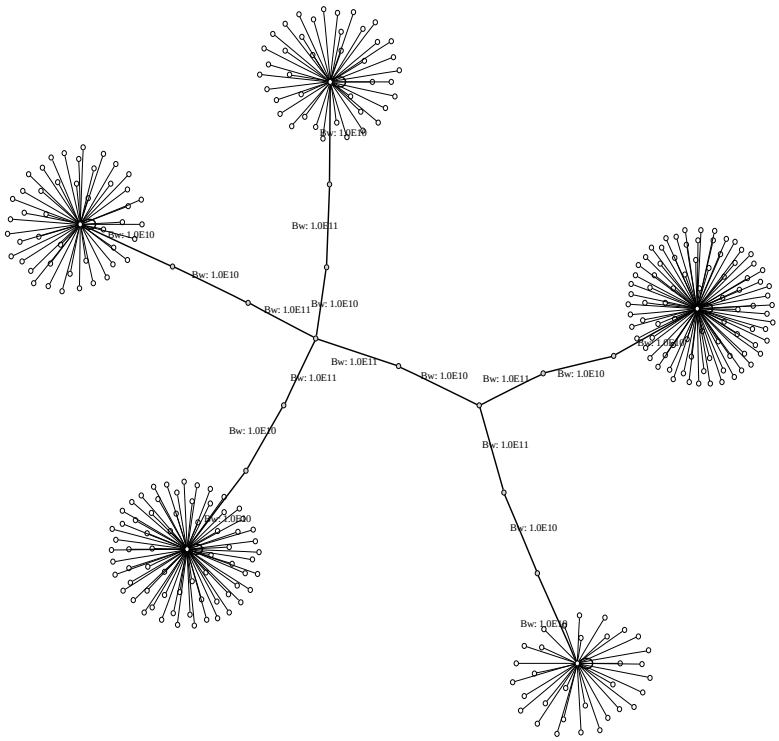
Relying on simulation during the design of parallel and distributed applications or as part of a validation process assumes that the simulations are driven using realistic platform configurations. This in turn requires to describe the simulated platform in a certain formalism that needs to respect the following properties :

Expressiveness The formalism should not be limited to the only resources simulated by the environment but also allow to declare additional arbitrary data that can be useful to end users.

[6] `http://www.cs.vu.nl/das3/`
[7] http://www.grid5000.fr

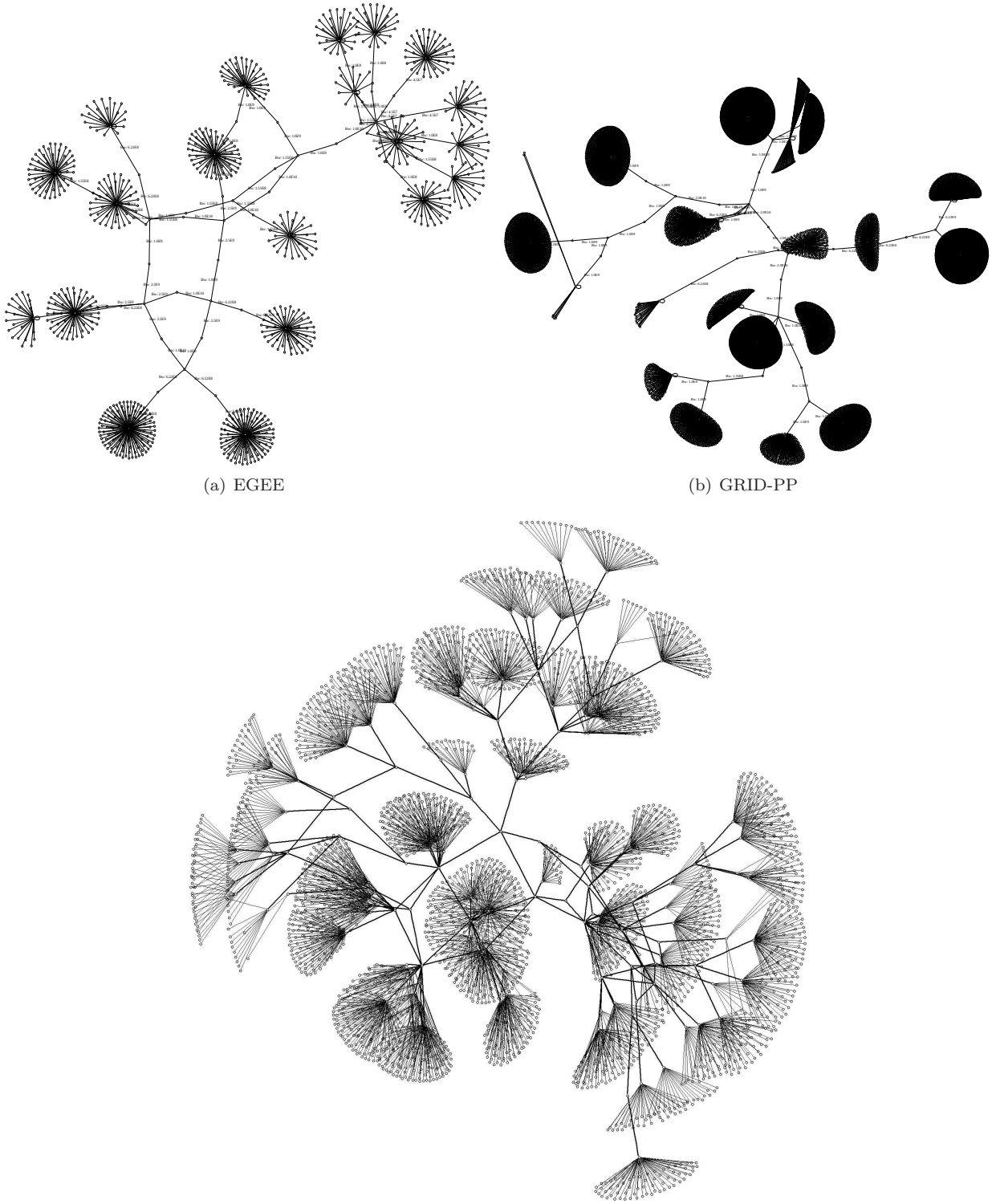


(a) Grid'5000.



(b) DAS-3.

Figure 1: Representations of the Grid'5000 and DAS3 experimental grid platforms.



(a) EGEE (b) GRID-PP

(c) LCG

Figure 2: Representations of the EGEE, GRID-PP and LCG production grid platforms.

Readability Platform description files should be easy to parse by computers but also easy to read or modify by human beings. This property allows users to export their files in or out their community.

Reusability It should be possible to use the same platform description file for several simulations.

Compactness Platform description files should be compact enough (up to hundreds of KB) to allow the generation of multiple files representing a large range of scenarios in a reasonable space.

In this report we showed that if each of the leading simulation environments has its own platform description formalism, none of it respects all the properties. As GridSim describes its platforms directly in the source code of the simulator, it clearly lack of reusability and also of readability as descriptions cannot be exported to another context. The text file of OptorSim is easy to parse but totally unreadable without mandatory explanation of the different fields. Both simulation toolkits are limited to small to medium scale platforms due to the chosen representation.

The platform description formalism of SimGrid in its 3.2 version also lacks of compactness and its expressiveness is limited to the simulated resources. Both issues were addressed in this report as we proposed to modify the DTD of the SimGrid toolkit to allow the description of large scale multi-cluster platforms through a more compact description of the compute resources and a factoring of the routing declaration. We also extended the language to add new features such as arbitrary properties, random generators and availability traces connection. Finally we modified the SimGrid XML parser to allow the handling of large files. Experiments showed that the new DTD leads to XML files more than 6,600 times smaller and 45 times faster to parse.

One remaining key issue is the memory usage made by SimGrid. As the whole routing table has currently to be stored into memory, the biggest platform that can be described with the proposed formalism comprises at most a few thousands of fully interconnected hosts. This memory limitation can be easily overcomed by compacting the routing table as we did in the XML file. This could easily be done by changing the way the `<cluster>` tag is handled.

As a future work we plan to export the proposed XML description of platform out of the SimGrid environment. We propose to build a catalogue of platform configurations comprising descriptions either inspired from real-world platforms or produced by random topology generators. We aim at offering the same kind of service as that proposed by the Grid Workload Archive[8]. We also plan to use the proposed formalism to add a wizard function in the deployment tool of the DIET[9] middleware. Finally we think about developing conversion tools between the SimGrid platform description formalism and those used by other popular simulation environments.

References

[1] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *International J. of High Performance Computing Applications*, 17(4), 2003.

[2] Rajkumar Buyya and Manzur Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *J. of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15), Decembre 2002.

[3] Henri Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, page 430, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[4] Henri Casanova, Arnaud Legrand, and Loris Marchal. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, may 2003.

[5] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*. IEEE Computer Society Press, March 2008.

[6] Kayo Fujiwara and Henri Casanova. Speed and Accuracy of Network Simulation in the SimGrid Framework. In *Workshop on Network Simulation Tools (NSTools)*, 2007.

[8] `http://gwa.ewi.tudelft.nl/`
[9] `http://graal.ens-lyon.fr/DIET`

[7] Dong Lu and Peter Dinda. Synthesizing realistic computational grids. In *Proceedings of ACM/IEEE Supercomputing 2003 (SC 2003)*, November 2003.

[8] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, Cincinnati, Ohio, August 2001.

[9] Martin Quinson. GRAS: a Research and Development Framework for Grid and P2P Infrastructures. In IASTED, editor, *International Conference on Parallel and Distributed Computing and Systems*, 2006.

[10] Al Kiswany Samer, Matei Ripeanu, Adriana Iamnitchi, and Sudarshan Vazhkudai. Are p2p data-dissemination techniques viable in today's data intensive scientific collaborations? In *European Conference on Parallel Computing (EuroPar), pp.404-414*, 2007. `http://www.ece.ubc.ca/~matei/PAPERS/europar2007.pdf`.

[11] Rich Wolski, Neil Spring, and Jim Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6):757–768, October 1999.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803