Synthesizing Generic Experimental Environments for Simulation

Martin Quinson
Nancy University/LORIA, INRIA
Nancy, France
Martin.Quinson@loria.fr

Laurent Bobelin Frédéric Suter
IN2P3 Computing Center, CNRS/IN2P3
Lyon-Villeurbanne, France
Frederic.Suter@cc.in2p3.fr

***Abstract*—Experiments play an important role in parallel and distributed computing. Simulation is a common experimental technique that relies on abstractions of the tested application and execution environment but offers reproducibility of results and fast exploration of numerous scenarios. This article focuses on setting up the experimental environment of a simulation run. First we analyze the requirements expressed by different research communities. As the existing tools of the literature are too specific, we then propose a more generic experimental environment synthesizer called SIMULACRUM. This tool allows its users to select a model of a currently deployed computing grid or generate a random environment. Then the user can extract a subset of it that fulfills his/her requirements. Finally the user can export the corresponding XML representation.**

I. INTRODUCTION

Experiments play an important role in Computer Science, especially in the field of parallel and distributed computing. It is the most common way to explore the combinatorial states of a theory to prove or disprove conjectures, validate a model in comparing its predictions with experimental results, or measuring the performance of a particular design under normal conditions. These mandatory experiments can be done following different methodologies going from executing a real application in a real environment (*in situ* experiments) to executing a model of an application on a model of an environment (*simulation*) [1]. Simulations may not be as realistic as *in situ* experiments but come with attractive features such as the reproducibility of results, an objective basis for application comparison, and the capacity to explore a broad range of experimental scenarios in a reasonable amount of time.

Figure 1 describes the different components of a simulation and their interactions. First, a simulator comprises an *application* to test, e.g., a peer-to-peer gossip protocol or a scheduling algorithm, and a *simulation kernel*. This kernel is the core of simulation toolkits such as GridSim [2], OptorSim [3] or SIMGrid [4]. Then a simulation implies the definition of an experimental scenario whose complexity may vary. A fundamental component is a *model of the experimental environment* or platform, i.e., an interconnection of computing elements through a network. The *input parameters* of the simulated application, e.g., the jobs to schedule, are also part of the scenario. The two remaining components add the capacity to inject external dynamic conditions that impact the application (the *workload*) or modify the infrastructure (the *availability changes*). During or at the end of a simulation run, the kernel can output several kinds of information. It can be raw data such as a trace of all the events that occurred. These data can also be post-processed to produce higher level information such as statistics or visualization.

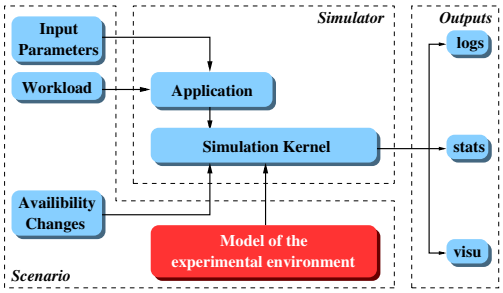

Figure 1. Different components of a simulation run and their interactions.

Depending on the simulation toolkit used, the simulator and the scenario are more or less tightly coupled. For instance a simulator written with GridSim also describes the experimental environment. With OptorSim, the network topology and the storage and computing elements are described in an input file. The applicative workload, i.e., the simulated jobs, is given in another file. Finally with SIMGrid, each component of the scenario is described in a distinct configuration file.

While the validity of the underlying models is the most critical issue for the *simulation kernel*, the success of a simulation campaign also depends on the diversity of the tested scenarios. The *input parameters* and the external *workload* are very dependent on the simulated application. Nevertheless it is possible to gather traces from real workloads. Some catalogs of generic traces exist, e.g., UMASS trace repository [5], but they often are specific to a given category of applications, e.g., Desktop Grid traces [6] or the Grid Workloads Archive [7]. Similar efforts have been conducted to gather realistic *availability* traces that impact the experimental environment. For instance the Failure Trace Archive [8] allows a simulator to take the volatility of the computing resources into account. The Load Trace Archive proposed by Dinda in [9] artificially changes the share of CPU available to the application.

In this paper we focus on the last component of a simulation scenario: the *experimental environment*. As for the other

components, many tools exist to synthesize realistic network interconnections [10], [11], complex computing resources [12] or combinations of both, i.e., computing grids [13], [14]. All these tools are specific to a given user community, i.e., networking, cluster, or grid computing. However simulation toolkits now target more research communities than these. For instance the SIMGrid toolkit has increased its scalability to handle peer-to-peer simulations, while GridSim has recently evolved into CloudSim [15]. Moreover new computing infrastructures such as Desktop Grids and Clouds become more prevalent. They come with new characteristics and requirements from a simulation point of view. Finally many grids are now deployed either for production or research purposes. Using descriptions of these grids in a simulation context could be of great help to application developers. Indeed simulation can be used in the development cycle to subject an application to controllable and reproducible execution conditions.

All these reasons have motivated the design of a synthesizer able to produce experimental environments suitable to any research community. The contributions of this paper are:

1) Clearly identify the requirements related to synthetic experimental environments expressed by different research communities in Computer Science (Section II).
2) Propose a tool called SIMULACRUM (Simulation pLAtform CReation and User-guided Modification). It relies on a modular generation process that can be adapted to the desired output (Section IV). SIMULACRUM uses some original mechanisms, i.e., promoters, labelers, filters, to go from a network model to a completely defined environment that perfectly matches the user's needs.

We also show the limitations of the existing tools in Section III. An evaluation of SIMULACRUM is given in Section V. Finally we conclude this work and present some future work in Section VI.

II. Requirements for a Generic Experimental Environment Synthesizer

In this section we distinguish two types of requirements. First we detail which kind of experimental environment is needed by which research community. We express the specificities of each type of environment through simple use cases. Then we give the desirable characteristics of a synthesizer of such environments for simulation purposes.

A. Types of Synthetic Environments

1) Networking: In the networking community simulation is used to assess the behavior and the efficiency of algorithms and protocols at differing scales, from local networks to the full Internet. The underlying synthetic environments thus have to reflect the fundamental properties of the actual structure of the Internet. Properties such as the presence of power-laws derive from empirical studies [16]. In this community a synthetic environment has to be fully connected and representative of real world networks. Moreover some qualitative information is also needed. For instance, link bandwidths and network delays can have a impact on the studied protocols. On the other hand, this community shows only little interest in the description of the computing resources located at the edges of the network.

2) Large Scale Distributed Systems: This domain covers the study and design of peer-to-peer algorithms, e.g., gossiping protocols or scalable data replication mechanisms. If the experimental environments needed by this community are as network-oriented as those required by the networking community, their characteristics are nevertheless different. Indeed the structure is here less important than the scale, typically hundreds of thousands entities. Moreover the main structural property is now the distance between the elements of the environment. This distance can be represented by the latency of the communication links. Recently the link bandwidth has gain some interest in peer-to-peer algorithm design. This characteristic then also has to be handled by a synthesizer. Finally some details about non-network resources may have to be described. For instance, the amount of available disk space that can be used by a peer-to-peer storage application.

3) Desktop Computing: Initiated by the SETI@Home project and generalized with the BOINC framework, Desktop Computing is now a common way to solve large scale computing problems. In this community, researchers rely on simulation to study fault-tolerance and scheduling algorithms to improve platform reliability and throughput. The same kind of experimental environments as in the previous use cases is needed, but the focus needs to be put on the computing power of each volunteering resource while the network capacities become less important.

4) Cluster computing: Simulation is also used in a High Performance Computing context, e.g., to compare the relative merits of different scheduling algorithms to manage commodity clusters [17], [18]. In opposition to the previous use cases, the network topology has here only little interest. Indeed batch schedulers usually manage one or a few clusters. The description and generation of the experimental environments are thus simpler. But this community also requires realistic and flexible descriptions. Ideally, it should be possible to users to gain access to descriptions of real production clusters. Adding flexibility to these descriptions allows users to easily study the behavior of their algorithms on smaller, larger, or upgraded versions of the initial cluster. A typical use case is to give some insights to decide what would be the most suited upgrade. Simulation can be used to replay some workload traces in different what-if scenarios, e.g., with twice as much processors or with a high performance network interconnect. Using less processors than available can also help to determine what would be the impact of switching off some machines (for maintenance or due to energy constraints).

5) Grid Computing: In computing grids, resources are often interconnected either through a private network or the infrastructure of National Research and Education Networks (NRENs). This is the case in production infrastructures such as the European grid EGEE (http://www.eu-egee.org/) which mainly relies on the GEANT network. GEANT is a pan-european network that interconnects european NRENs. Re-

Table I
REQUIREMENTS FOR SYNTHETIC EXPERIMENTAL ENVIRONMENTS PER RESEARCH COMMUNITY.

Community	Desired Topology	Computing Resources	Network Resources	Properties
Networking	Similar to Internet	(none)	Latency	LAN/WAN
Large Scale Distributed Computing	Large scale (not realism)	Single nodes	Latency, Bandwidth	Disk
Desktop Computing	Similar to Internet	Single hosts	Latency, Bandwidth	Computing power
Cluster Computing	Simple (flat)	Cluster	Latency, Bandwidth	Computing power
Grid Computing	Simple (hierarchical)	Cluster	Latency, Bandwidth	Power, Services
Cloud Computing	Similar to Internet	Cluster	Latency, Bandwidth	Power, Storage, Services

search grids such as the french initiative Grid'5000 (http://www.grid5000.fr) also have their own private network. On this platform, the private backbone network infrastructure is provided by the French NREN. This leads to less complex network topologies than what can be found on the Internet. In NRENs or networks of NRENs, end-to-end paths are shorter, and backbones are usually made of high bandwidth.delay optic fiber links. Moreover the compute resources typically deployed in Grids are commodity clusters.

There is also a strong need for resource descriptions in this research field. Indeed many scheduling algorithms perform resource match-making to map jobs. Various properties, such as the OS, the available memory and disk space, which libraries are installed, are part of the experimental environment. Such properties are usually not handled by simulation toolkits. Then they may be seen as a distinct part of a simulation scenario. Nevertheless this information is tightly coupled to the environment and has to be handled by a generic synthesizer.

Another important requirement in the domain of grid computing is to run simulations on an experimental environment as close as possible to reality. Ideally, descriptions of deployed grid infrastructures should be available. In a production context, this would help application developers to prepare an experiment campaign, e.g., fixing the input parameters, in a controlled but realistic setting. Another possible use of simulation is to replay in a simulation context an already executed experiment. As for cluster computing, some performance assessments can be done by introducing some variations in the experimental environment.

6) Cloud computing: The emerging field of Cloud Computing can been seen as an hybrid of Grid and Desktop computing. In terms of experimental environments, this means the combination of their respective requirements. Indeed a cloud is composed of powerful compute and storage resources such as clusters, but accessed through the Internet. Moreover the accurate description of the resources and services available on each site is mandatory. Simulation in the domain of Cloud Computing can help to the deployment of a service infrastructure in an efficient way.

Table I summarizes the requirements in terms of network topology, type of computing and network resources, and additional properties, expressed by the different research communities considered in this section.

B. Requirements for a Synthesizer

We distinguish two categories of requirements. First we present those related to the produced experimental environments. Then we describe what should be the features of the tool itself. Note that a subset of these requirements has been identified by the authors of BRITE in [11].

The requirements about the produced synthetic environments have already been identified in the previous section. We just recall the two main targeted characteristic that a synthesizer must ensure. The first one is the *scalability*. This means the capacity to produce large scale experimental environments in a reasonable time. The second essential characteristic is *Representativeness*. The produced environment, and not only the network topologies, has to reflect many aspects of the real infrastructures targeted by researchers in Computer Science.

To satisfy all the requirements expressed by the different communities, a experimental environment synthesizer first has to be *generic*. This implies a good *modularity* of the tool to easily adapt the generation process to the needs. A similar feature, outlined in [11], is the *Inclusiveness*. A good synthesizer has to combine the strengths of as many generation models as possible. It also has to be *extensible*, i.e., new features must be easy to add. The *interoperability* is another important feature to target as many simulation toolkits as possible. As any good software product, a experimental environment synthesizer also has to be *efficient* in terms of CPU and memory consumption, *robust*, and include some error detection capacities.

A final set of requirements is related to the user. To be widely used, an experimental environment synthesizer has to be *user-friendly*. It implies a fast learning curve and a simple interface. Moreover, the user has to be involved in the generation process. This *user control* is not only necessary at the beginning of the process but all along its different steps in an interactive way.

III. RELATED WORK AND LIMITATIONS

Several tools have been proposed over the last decade to synthesize experimental environments. As explained in the previous section, the desired environments rely on a network topology and compute resource models. Tiers [10] follows a top-down approach to generate a N-Level topological graph. It starts from a connected graph and replace a node by another connected graph at each step. Tiers also adds a semantic to the edges of the graph to distinguish LAN, MAN and WAN networks. Unfortunately, this semantic is not exploited

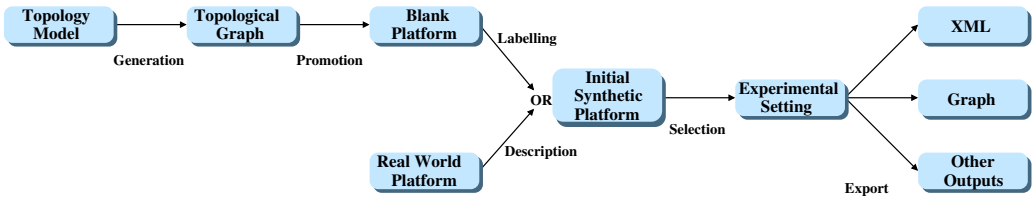

Figure 2. Generation flowchart of the SIMULACRUM tool.

to derive network link latency and bandwidth values. Moreover Tiers does not consider non-network resources. Tiers can thus only be used to create an initial topological graph.

The BRITE tool [11] provides a unified framework for the generation of network topologies. A particular emphasis on topologies reflecting the structural properties of the Internet, e.g., hierarchical structure and degree distribution, is given. As in Tiers, the computing resources located at the edges of the network are ignored by BRITE. Nevertheless it allows users to label communication links with bandwidth values. A flaw of the generation process of BRITE has to be underlined. As mentioned in the previous section, connectivity is an essential feature in the networking community. When one of the network models used by BRITE produces a non-connected graph, edges are added to reach full connectivity. The flaw comes from the way these edges are added as it does not necessarily respect the initial distribution rule. A better solution would be to generate a new graph until a connected graph is produced.

While the previous tools focus on network topologies, some work exists about the generation of synthetic computing resources. From the observation that grids are principally made of clusters, a commodity cluster synthesizer has been proposed in [6]. The authors examined 114 production clusters comprising more than 10,000 processors in terms of processor architecture, clock frequency, cache size, number of processors, network interconnect, disk capacity, or release date. They came up with statistical models to allow users to extrapolate for future configurations. These models have been validated against an other set of clusters. By contrast with Tiers and BRITE that generate network topologies with no computing resources, this commodity cluster synthesizer can produce multiple cluster-like resources, but does not interconnect them. Then the experimental environment is incomplete for some targeted communities. Furthermore this synthesizer does not include information such as computing power (measured in classical units such as Gflop/s, MIPS, or SPECint), which is a fundamental information for simulation kernels. Finally, the tool itself was never publicly released.

The GridG [14] project is a computational grid synthesizer. To the best of our knowledge it is the only tool that synthesizes experimental environments with computing resources connected through a network. It relies on structured topologies (obtained with Tiers) that follow the out-degree power-law. The routers, hosts, and links of the produced topologies are then annotated with attributes such as memory size, number of CPUs, disk size, or bandwidth. The GridG annotation mechanism also supports user-supplied conditional probability rules to define correlations among the attributes. A main drawback of GridG is the limitation to a single kind of network topology. Although this model is hierarchical and follows power-laws as in the Internet, users of GridG cannot test their algorithms or protocols with other models. GridG is then not suited to the networking community needs. Moreover it has be shown that power-laws are not sufficient to represent the Internet [19], [20]. This limitation is then even more problematic.

Finally a first attempt to create a compendium of sub-platforms that covers a broad range of characteristics from a real-world compute grid was proposed in [21]. This work is not really a synthesizer, but addresses the need for the description of actually deployed computing infrastructures. Its main drawback is that the subset selection is not user-driven.

IV. THE SIMULACRUM TOOL

As shown in Section II, each research community has its own requirements for the experimental environments. A generic synthesizer thus has to cover all these requirements. Our proposed synthesizer follows the generation flowchart depicted in Figure 2 to achieve this objective. Note that some steps may be only no-ops depending on the expected output.

One of the first original feature of SIMULACRUM is that its generation flowchart has two entry points. Descriptions, in the XML format proposed in [13], of currently deployed grid infrastructures are provided, as needed by cluster and grid communities. In the current version, descriptions of Grid'5000 and DAS-3 (http://www.cs.vu.nl/das3/) are provided.

In what follows we detail the different steps of this flowchart following the longest path. This generation process mainly concerns the networking, large scale distributed, desktop computing, and cloud computing research communities.

1) Topological Graph Generation: To create a topological graph, SIMULACRUM relies on several models. Classical topologies such as ring, star, or clique are of course available. Moreover SIMULACRUM implements models that spread the nodes over a unit-square area and connect two nodes u, v with a probability $P(u, v)$ following different distributions such as uniform, exponential, Waxman [22], and Zegura [23]. The topologies produced by these models are flat in opposition to those produced by hierarchical generators such as Tiers [10]. Such hierarchical configurations are not achievable in SIMULACRUM so far, but we plan to implement a specific promoter changing a node into a full platform. Finally, SIMULACRUM provides another class of interconnection

generators encompassing degree-based models such as the one proposed by Baràbasi and Albert in [24]. This model, based on incremental growth of the platform and affinity connexion, is known to better follow the power-laws.

In Section III we have outlined a flaw in the way BRITE produces connected graphs with either the Waxman or Baràbasi-Albert models. To respect the initial definitions of these models SIMULACRUM calls the generation function until the produced graph is connected. If after 10 tries, no connected graph can be generated, the user is asked to modify the generation parameters.

At the end of this first step, SIMULACRUM manages a connected topological graph composed of abstract nodes. These nodes have no particular type yet. Moreover the edges of this graph only represent the fact that two abstract nodes are connected or not. This graph just gives the structure of the experimental environment. The next two steps aim at adding qualitative information to these abstract entities.

2) Node Promotion: The second step in our generation flowchart consists in converting the abstract nodes of the topology graph into computing (hosts and clusters) and networking (routers) resources. The difficulty here is to express complex decision-making processes such as "change one half of the graph leaves into low-cost desktop machines and the second half into small clusters; nodes with medium degree should be changed into powerful computational servers; nodes with high degree should remain routers".

The original approach followed by SIMULACRUM is to define a chain of *promoters*, which expresses the transformation of the topology graph into an interconnection of resources. A promoter is a decision-making rule encompassing a filter (deciding to apply this promoter to a given node or not) and a generator (in charge of promoting the node when the rule applies). Note that only the nodes of the topological graph are affected by this modification. The edges still only express whether two resources are connected or not.

For each promotion rule, several filtering patterns are available. They can be combined in a logical AND manner to express several properties to respect. Some filters depend on the properties of the node, e.g., its degree, while others depend on the targeted "blank platform". For instance, a filter may be applied while the number of computing resources is under a certain threshold.

If a node of the topological graph gets caught in the filter of a promotion rule, it is promoted to the corresponding resource type. A node can be changed into a single host, i.e., a desktop computer characterized by its compute speed (in Gflop/s), or an homogeneous cluster, i.e., multiple hosts interconnected through a LAN. For both types of promotions, the characteristics of the target resource can be fixed by the user or picked uniformly within an interval. The nodes that are not selected by any filter remain routers.

Note that the promoters are considered in order for each node. The first promotion rule whose filter catches a node is applied to it and the subsequent promoters are then skipped for this already promoted node. The decision-making process introduced above can be informally expressed by the following chain of promoters.

Promoter 0: AND(is leaf, probability 0.5)
⇒ small desktop
Promoter 1: node is leaf
⇒ small cluster
Promoter 2: $degree \in [2, 5]$
⇒ powerful server (other nodes remains routers)

As expressed in Section II, it is often useful to add arbitrary properties to the platform elements (represented as $key \times value$ couples). To that extent, a list of *property adders* is associated to each promoter. Each of these adders is in charge of adding a given property to any node created by this promoter. The value is either a given string, or a numerical value picked uniformly in an interval. This allows for the description of services and data storage components that are typical in cloud computing.

Adding other filters or promoters would be straightforward thanks to the tool modularity. To do so, one would only have to write a new Java class in less than 50 lines of source code, and add an instance of it into a specific array.

At the end of this second step all the abstract nodes of the topological graph have been changed into a compute (hosts and cluster) or network (routers) resource, and user-defined properties were added to these nodes. We thus have a "blank platform." It remains to convert the edges representing the connections into communication links.

3) Edge Labelling: Once each node of the topological graph has been promoted into its final type of resource, communication properties, e.g., latency and bandwidth, still have to be associated to the edges of the topological graph. SIMULACRUM relies on the same promotion mechanism as for the nodes by using a chain of *edge labelers*. These rules are also applied in order, in an exclusive manner, and some properties adders can be associated to each labeler.

The currently available filters act on the length of the edges. The length of an edge is defined as the Euclidean distance between the nodes it interconnects on the unit-square area. When an edge is caught in a filter, it becomes a communication link that is labeled with a latency, a bandwidth and a sharing policy. The values of the first two labels can either be fixed by the user or uniformly picked within an interval. The sharing policy models hubs and switched networks. It expresses whether or not a communication link will suffer from contention.

As for the nodes, adding new edges labeling filters is possible. For instance, it would be interesting to allow filtering on the amount of network paths traversing a given link, to label the most used links as backbone elements (large bandwidth, small latency) while other ones links would be labeled as network edges.

At the end of this step, SIMULACRUM handles a generated synthetic platform described in the same way as real-world computing grids. The next section describes how a user of SIMULACRUM can specify which subsets are of interest.

4) Subset Selection: SIMULACRUM provides two ways to select a subset of the initial synthetic platform, be it generated or a description of an existing computing grid. First a user

Table II
EXAMPLES OF SYNTHESIZER SETTINGS FULFILLING THE NEEDS OF EACH RESEARCH COMMUNITY.

Community	Topology Model	Real world description	Node Promotion	Labelling	Properties
Networking	Barabàsi	Internet-like	fixed host	latency from distance	WAN/LAN
Large Scale Distributed Computing	Waxman/Zegura	Large scale	fixed host	fixed characteristics	node
Desktop Computing	Barabàsi	Internet-like	uniform host	lat&bw from dist.	disk
Cluster Computing	only one cluster	flat tree	uniform cluster	none	disk
Grid Computing	Waxman/Zegura	hierarchical	uniform clusters	lat&bw from dist.	disk, services
Cloud Computing	Barabàsi	Internet-like	uniform clusters	lat&bw from dist.	disk, services

can manually discard some of the hosts or clusters composing the initial platform. For each cluster it is also possible to modify the number of hosts composing it by providing a new regular expression. Note that this modification can decrease or increase the number of computing resources and alter the properties of the generation model. During the selection of resources, the user is notified of the evolution of the characteristics of the experimental environment. This way the user can continue to modify the platform until the desired resource heterogeneity is reached.

To obtain all the subsets of the initial synthetic platform that satisfy certain properties, SIMULACRUM also provides an automatic selection mechanism. This interactive process allows the user to express the different properties that a subset must meet as a chain of *filters*. At the end of the selection process, the list of all the subsets that passed through all filters is displayed.

Several filters are available. Some of them consider the structural properties of a subset, i.e., the number of nodes in the topological graph, the number of hosts or clusters, or the diameter of the graph. Another class of filters allows the user to characterize the statistical distribution of the hosts' compute speed within a subset [25]. Note that the compute speed absolute value is less relevant than the ratio between the highest and lowest compute speeds to determine the heterogeneity of a given subset. Thus we compute the statistical moments over the logarithm of the compute speed.

The *distribution support* filter fixes the interval within which the compute speed of each node of the subset must lie. The *average* filter ensures that the compute speed average remains within the given interval. The *variance* and *standard deviation* filters help to control whether the compute speeds are concentrated around the average or evenly distributed between the extrema. The *skewness* filter corresponds to the third standardized moment. It measures the asymmetry of the probability distribution. A negative value indicates that the mass of the distribution is concentrated on higher values with relatively few small values. The average is then bigger than the median. A positive value indicates the contrary. Finally the *kurtosis* filter corresponds to the fourth standardized moment, which measures the "peakedness" of the distribution. A high kurtosis means that most of the variance is due to infrequent extreme deviations, while in flatter distributions the variance comes from frequent but modestly-sized deviations.

Such a filtering process implies an exhaustive search among 2^n possible combinations where n is the number of distinct compute resources of the initial platform. Several techniques can be used to reduce the cost of this search. For instance an homogeneous cluster can be considered as a single entity instead of treating each of the hosts it comprises separately. Moreover the structural filters have to be applied uppermost to reduce the space search for the application of the statistical filters. In addition, the search is stopped as soon as 100 candidate platforms fulfilling all the filters are found. The search can be interrupted and is conducted in a separated thread to prevent any user interface "freeze".

As for the promotion and labelling mechanisms, it is possible to add new filters to SIMULACRUM by writing specific classes being about 25 lines of source code (not counting the actual filtering code). For instance other graph theory statistics should be considered, such as the graph resilience, distortion, the distribution of the degrees, or a metric measuring whether the graph is scale-free introduced in [20]. Then, the distribution of the network path communication abilities (bandwidth and latency) could be filtered similarly to the compute speed distribution.

We now have a completely defined experimental environment. This description is ready to be used as part of a simulation scenario. In Section II, we expressed the needs of each research community regarding experimental environments for simulation studies. Table II exemplifies some SIMULACRUM settings that correspond to these needs. For example, since Grid and Cloud researchers are mainly interested in interconnections of clusters, they should promote nodes into clusters (with either fixed capacities or uniformly picked ones). Moreover the interconnection between clusters is of little interest in Grid community. A classical generator such as Waxman or Zegura is then sufficient. To mimic a Cloud in which the clusters are connected directly to the Internet, the Barabàsi topology generator will be preferred since the produced topologies are more similar to the Internet structure. In both cases, selecting subsets of existing platforms based on the computational power distribution is also an interesting approach. In contrast, large scale distributed platforms should probably be generated using any method and then filtered on graph metrics such as the diameter.

5) Export: The final step of the generation process allows users to make some final adjustments. The current version of SIMULACRUM exports an XML representation of the produced experimental environments or graphically displays

the platform using the classical `dot` tool. Note that this representation can be edited within the SIMULACRUM GUI to allow users to make final adjustments at will. This representation is based on the XML format described in [13]. In this section we briefly detail the tags corresponding to hosts, compute clusters and communication links.

The `<host>` tag describes a compute resource which has an `id` and runs at a certain compute speed (or `power`). The following example describes an host named HOST1 that computes 1 billion of floating operations per second.

```
<host id="HOST1" power="1E9"/>
```

In this XML format network links represent one-hop network connections. They are characterized by their `id`, `bandwidth` and `latency`. The following example declare a network link named LINK1 having bandwidth of 10 Gb/s and a latency of $10\,ms$.

```
<link id="link1" bandwidth="1.25E9"
      latency="1.0E-4"/>
```

By default a network link is shared, i.e., if more than one flow go through a link, each gets an equal share of the available bandwidth. Conversely if a link `sharing_policy` is set to `fatpipe`, each flow going through this link will get all the available bandwidth, whatever the number of flows.

The `<cluster>` tag defines an homogeneous cluster of n processors, each of them being connected to a common backbone by a private link. The backbone is in turn connected to the outer world. The name of the backbone is obtained by appending "_bb" to the cluster name while the private links are named after the host they serve.

```
<cluster id="cluster1" prefix="NODE-"
         suffix=".CLUSTER.FR"
         radical="1-3" power="4.311E9"
         bw="1.25E8" lat="1E-4"
         bb_bw="1.25E9" bb_lat="1E-4"/>
```

This tag defines an homogeneous cluster of 3 hosts whose names are NODE-X.CLUSTER.FR, with $X \in \{1, 2, 3\}$. Each host runs at 4.311 Gflop/s and is connected to a private link having a bandwidth of 1 Gb/s and a latency of $10\,ms$ (given by the **bw** and **lat** attributes). Each pair of hosts is connected through their private links and an internal backbone whose bandwidth is 10 Gb/s and latency is $10\,ms$ (given by the **bb_bw** and **bb_lat** attributes). By default this internal backbone link follows the `fatpipe` sharing policy.

Once the user is satisfied with the XML description, he/she can save the corresponding file. This file becomes a part of the simulation scenario. It can be directly used as input of a SIMGrid simulator. We plan to develop conversion tools towards other simulation toolkit formats. Such add-ons will increase the interoperability of SIMULACRUM.

V. EVALUATION

In this section we evaluate the generation capacities of SIMULACRUM according to the following three metrics:

a) Generation time: First we measure how much time it takes to generate a synthetic experimental environment depending on the targeted research community (and thus to a certain set of requirements). Timings were obtained on a Intel Core2 X7900 computer at 2.80GHz. The JVM were configured to use up to 2Gb of heap space.

b) Size of output file: We also measure the size of the XML files to assess their compactness (and thus usability).

c) Generation limits: Here we determine what is the size of the biggest instance of a given type of environment we can generate with SIMULACRUM. This limit (mainly due to the memory availability) varies depending on requirements such as the type of network connections or obviously the number of nodes in the topological graph.

For our evaluation we consider three types of experimental environments: a Desktop Grid, a multi-cluster (or grid) and a Cloud. The SIMULACRUM settings for each of these environments were given in Table II. More precisely, the Desktop Grids were generated by interconnecting the given amount of hosts with the Baràbasi algorithm. For multi-clusters and clouds, the nodes are promoted to clusters. The Waxman algorithm (with $\alpha = \beta = \frac{1}{2}$) were used to interconnect the multi-cluster platforms while Baràbasi were used for clouds. The settings used for desktop grids and clouds lead to better performance because of the relatively large amount of links generated by the Waxman algorithm with these parameters.

Table III
GENERATION TIME AND OUTPUT SIZE PRODUCED BY SIMULACRUM FOR DIFFERING SIZES OF DESKTOP GRIDS, MULTI-CLUSTERS, AND CLOUDS.

# nodes	Desktop Grids		Multi-clusters		Clouds	
100	0.6s	26kb	2s	500kb	0.8s	45kb
500	2s	140kb	24s	12Mb	4s	230kb
1000	5s	280kb	215s	46Mb	7.7s	460kb
2000	20s	560kb	(out of memory)		26s	930kb
3000	50s	830kb	(out of memory)		(out of memory)	

We have also investigated whether it is possible to extract subsets of the Grid'500 platform as described in [21]. To this end, we searched for subsets of the platforms that contained exactly 8 clusters and so that the heterogeneity, defined as the maximum ratio of the CPU powers in two distinct clusters, is no larger than 1.1. It took SIMULACRUM 20 seconds to identify 11 such subsets out of the 4194201 possible configurations. For comparison purposes, in [21], the authors found 10 such subsets (based on the Grid'5000 platform configuration in 2007).

VI. CONCLUSION AND FUTURE WORK

In the field of parallel and distributed computing, simulation is a common way to performs experiments to validate a protocol or compare several algorithms under various conditions. In this context the variety, the representativity and the realism of the simulation scenarios are important characteristics to take into account. This is particularly true for the model of the experimental environment onto which the behavior of the simulated application is tested. q Choosing the right

experimental settings for a given study is a methodological issue faced by every researcher studying distributed systems through simulation, but the criteria of platform settings quality differ with the research communities. For instance, in networking studies, the topological characteristics of the network are crucial while the nodes at the edges can be neglected. On the contrary, Grid and Cluster researches are more focused on quantitative differences in terms of communication and computation capacities than on the qualitative characteristics of the interconnections.

These differences explain why existing synthesizing tools are often dedicated to a specific community. In this paper we have presented the SIMULACRUM (Simulation pLAtform CReation and User-guided Modification) tool which produces, in interaction with the user, generic synthetic experimental environments. It combines the models and approaches found in existing tools to original features such as arbitrary properties definitions. Contrary to existing synthesizers, SIMULACRUM is not limited to a specific community. A very promising feature is its ability to select subsets of existing platforms based on user-defined filters. It allows for a double-check of the characteristics of the selected platforms. For instance, it could help to understand the performance variations observed during the experiment in light of these inherent experimental settings' characteristics.

The SIMULACRUM project is part of the Platform Description Archive (PDA) which is an effort to make platform descriptions and tools available to users of simulation toolkits. The main goal of this archive is to ease the reproduction of already published simulation results by sharing the used experimental environments. SIMULACRUM can be downloaded from the home page of the PDA: http://pda.gforge.inria.fr.

In future work, we plan to further enrich the tool by adding more generation methods, promoters and filters such as a hierarchical generation method or other graph-based metrics such as resilience, scale-freeness or betweenness. We also plan to provide a way to interactively visualize and edit the selected platforms, as well as exporters to other simulation toolkits and file formats. Finally, we plan to couple this tool to the SimGrid toolkit to allow the users to specify and run the complete experiment campaign within SIMULACRUM.

ACKNOWLEDGMENT

The authors are indebted to several colleagues for ongoing conversations and valuable feedback, especially Arnaud Legrand and Henri Casanova. This work is partially supported by the ANR project USS SimGrid (08-ANR-SEGI-022).

REFERENCES

[1] J. Gustedt, E. Jeannot, and M. Quinson, "Experimental Methodologies for Large-Scale Systems: A Survey," *Parallel Processing Letters*, vol. 19, no. 3, pp. 399–418, Sep. 2009.

[2] 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: Practice and Experience*, vol. 14, no. 13–15, pp. 1175–1220, Dec. 2002.

[3] W. Bell, D. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini, "OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies," *International Journal of High Performance Computing and Applications*, vol. 17, no. 4, pp. 403–416, 2003.

[4] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *10th IEEE International Conference on Computer Modeling and Simulation*, Mar. 2008.

[5] T. Wood, "UMASS Trace Repository." [Online]. Available: http://traces.cs.umass.edu

[6] D. Kondo, A. Chien, and H. Casanova, " Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids," in *Proceedings of SuperComputing'04*, Pittsburgh, PA, Nov. 2004.

[7] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema, "The Grid Workloads Archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, Jul. 2008.

[8] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010)*, Melbourne, Australia, May 2010.

[9] P. Dinda, "The Statistical Properties of Host Load," *Scientific Programming*, vol. 7, no. 3-4, pp. 211–229, Nov 1999.

[10] K. Calvert, M. Doar, and E. Zegura, "Modeling Internet Topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–168, Jun. 1997.

[11] A. Medina, A. Lakhina, I. Matta, and J. 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, Aug. 2001.

[12] Y.-S. Kee, H. Casanova, and A. Chien, "Realistic Modeling and Synthesis of Resources for Computational Grids," in *Proceedings of ACM/IEEE SuperComputing 2004 (SC'04)*, Pittsburgh, PA, Nov. 2004.

[13] M.-E. Frincu, M. Quinson, and F. Suter, "Handling Very Large Platforms with the New SimGrid Platform Description Formalism," INRIA, Technical Report 0348, Feb. 2008. [Online]. Available: https://hal.inria.fr/inria-00256883

[14] D. Lu and P. Dinda, "Synthesizing realistic computational grids," in *Proceedings of ACM/IEEE Supercomputing 2003 (SC'03)*, Nov. 2003.

[15] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities," in *Proceedings of the 7th High Performance Computing and Simulation Conference (HPCS)*, Leipzig, Germany, Jun 2009.

[16] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," in *ACM Annual Conference of the Special Interest Group on Data Communication (SIGCOMM)*, Cambridge, MA, Sep. 1999, pp. 251–262.

[17] Y. Caniou and J.-S. Gay, "Simbatch: an API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems," in *Proceedings of the IEEE Workshop on Secure, Trusted, Manageable and Controllable Grid Services (SGS)*, Las Palmas de Gran Canaria, Aug. 2008.

[18] D. Klusácek, L. Matyska, and H. Rudová, "Alea - Grid Scheduling Simulation Environment," in *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM'07)*, ser. LNCS, vol. 4967, Gdansk, Poland, Sep. 2007, pp. 1029–1038.

[19] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "The Origin of Power-Laws in Internet Topologies Revisited," in *Proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, New York, NY, Jun. 2002.

[20] L. Li, D. Alderson, J. C. Doyle, and W. Willinger, "Towards a Theory of Scale-Free Graphs: Definition, Properties, and Implications," *Internet Mathematics*, vol. 2, no. 4, pp. 431–523, Jan. 2005.

[21] F. Suter and H. Casanova, "Extracting Synthetic Multi-Cluster Platform Configurations from Grid'5000 for Driving Simulation Experiments," INRIA, Technical Report 0341, 2007. [Online]. Available: https://hal.inria.fr/inria-00166181

[22] B. Waxman, "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[23] E. Zegura, K. Calvert, and M. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 770 – 783, Dec. 1997.

[24] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, pp. 509–512, Oct. 1999.

[25] W. Encyclopedia, "Descriptive statistics." [Online]. Available: http://en.wikipedia.org/wiki/Descriptive_statistics