# Experimental Validation of Grid Algorithms: a Comparison of Methodologies

Emmanuel Jeannot
INRIA Nancy Grand-Est
LORIA, Nancy University, CNRS
Emmanuel.Jeannot@loria.fr

## Abstract

*The increasing complexity of available infrastructures with specific features (caches, hyperthreading, dual core, etc.) or with complex architectures (hierarchical, parallel, distributed, etc.) makes models either extremely difficult to build or intractable. Hence, it raises the question: how to validate algorithms if a realistic analytic analysis is not possible any longer? As for some other sciences (physics, chemistry, biology, etc.), the answer partly falls in experimental validation. Nevertheless, experiment in computer science is a difficult subject that opens many questions: what an experiment is able to validate? What is a "good experiments"? How to build an experimental environment that allows for "good experiments"? etc. In this paper we will provide some hints on this subject and show how some tools can help in performing "good experiments". More precisely we will focus on three main experimental methodologies, namely real-scale experiments (with an emphasis on PlanetLab and Grid'5000), Emulation (with an emphasis on Wrekavoc:* `http://wrekavoc.gforge.inria.fr`*) and simulation (with an emphasis on SimGRID and GridSim). We will provide a comparison of these tools and methodologies from a quantitative but also qualitative point of view.*

## 1 Introduction

Computer Science is the science of information. As stated by Peter J. Denning et al. in [9], "*The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis design, efficiency, implementation and application*". There are several ways to perform such study. A first approach is to classify knowledge about information through an analytic work using the mathematics as a tool. Such an approach as been proved very efficient but is limited when a studied object becomes very complex and its understanding is hard to achieve using models. In the case where the objects (hardware, program, algorithms, etc.) are hard to understand analytically, a second approach, based on experiments allows to gather and classify some knowledge through observations [6].

Modern systems such as network, computers, programs, algorithms are more and more complex. Hence, their analytical understanding becomes more and more difficult and the experimental approach more and more necessary to understand, test, validate or compare such objects. This is even more important in the case of grids. Indeed, grids are built using computers that have very advanced features (caches, hypethreading, multi-core, etc.), which use operating systems that embed state-of-the-art solutions such as process scheduling, virtual memory, thread support. On top of the OS, runtime and middelware environments play a very important role in the overall performance and even different implementations of the same standard can impact the behavior. Moreover, the fact that, in a grid, the different resources can be heterogeneous, hierarchical, distributed or dynamic makes the picture even more complex.

If the role of experiments becomes more and more important in computer science and especially in grid computing, it is noticeable that the experimental culture of computer scientists is not comparable to other scientists in fields such as biology or physics. For instance, a study led by Luckovicz et al. [14] in the early 90's and taken up again by Tichy [17] shows that among computer science published articles in ACM journals, between 40% and 50% of those that require an experimental validation had none. The same study shows that this ratio falls to 15% for journals

in "*optical engineering*". A statistic study published in 1998 [19] on the experimental validation of results in computer science on more than 600 articles published by the IEEE concludes similarly that "*too many articles have no experimental validation*" even if quantitatively this proposition tends to decrease with time (the study covers the years 1985, 1990, 1995). These elements show that, in computer science, the experimental culture is not at the same level than in other sciences, even if it is improving with the time (at least quantitatively). This shows that we lack methodologies and tools to conduct experiments.

The goal of this article is to discuss and study the importance of experiments in computer science. We focus on large-scale systems and grids but large part of what is written here is general enough to be applied to any other fields of computer science. Hence, we will first discuss the role of experiments and the properties an experiment must fulfill (section 2). In Section 3 we will then discuss the different possible methodologies to conduct an experiment. A survey and a comparison of the experimental tools in the case of distributed computing and grid will be done in Section 4. Finally, we will conclude in Section 5.

# 2 Role of experiment in computer science

## 2.1 Related work

The question of computer science as an experimental science is not new but has recently raised new concerns.

The question of computer science as a science is brightly and positively answered in [8]. The point is that, even if there are some disagreements between computer scientists on whether computer science is science (or technology, or engineering, or hacking, etc.), there is no doubt that a large part of the discipline is science (which is not contradictory with the fact that computer science is also engineering). In [6] and [7] Denning discusses on the importance of experiment in computer science. For instance, he highlights that the LRU paging algorithm has been proved better then the FIFO strategy experimentally. As stated in the introduction the lack of experiment in computer has been emphasized in [14, 17]. In [11], D. G. Feitelson shows that experiment in computer science should not be so different than in other science, but there is not a real experimental culture in our discipline. In [10], Johnson discusses the issues of experimental analysis of algorithms. The author presents some principles that should govern such analysis from the novelty of the experiments to the presentation of the results.

In the following we will discuss and present in details the above issues.

## 2.2 Role of experiments

Denning et al. in [9] present three paradigm of the discipline of computing: theory, abstraction (modeling) and design. For each of these paradigms, the authors show that there is a feedback loop that allows to iterate the process and therefore enables progress and improvement of the object that one seek to construct (*i.e.* a theory, a model or a program). These three feedback loops are also used by Feitelson in [11] and are shown in Figure 1. We first want to emphasis that these three methodologies are complementary. Let us take the example of the design of an algorithm. The analysis of this algorithm must use the modeling of the reality that has to be experimentally validated. With the use of mathematics it is possible to demonstrate some property of this algorithm (complexity, approximation ratio, etc.). Moreover, an implementation of this algorithm will allow to test on real cases its efficiency.

As shown in this example and in the Fig. 1, one can remark that it is experiments and tests that enable progress for the modeling and the design paradigm. Therefore there are two types of experiments:

- a first category allows to validate a model in comparing its prediction with experimental results;
- a second category allows to validate the design quantitatively (*i.e.* by measuring the performance of design under normal conditions) .

However, it is important to understand that these two validations can also occur at the same time. For instance the validation of the implementation of an algorithm targets that both the grounding modeling is precise and that the design is correct.

Experiments have also an other very important role. In [17] Tichy presents the main advantages of experiment: in testing hypothesis, algorithms or programs experiment can help to construct a database of knowledge on theories, methods and tools used for such study. Observations can also lead to unexpected or negative results and therefore eliminate some less fruitful field of study, erroneous approaches or false hypothesis. Therefore

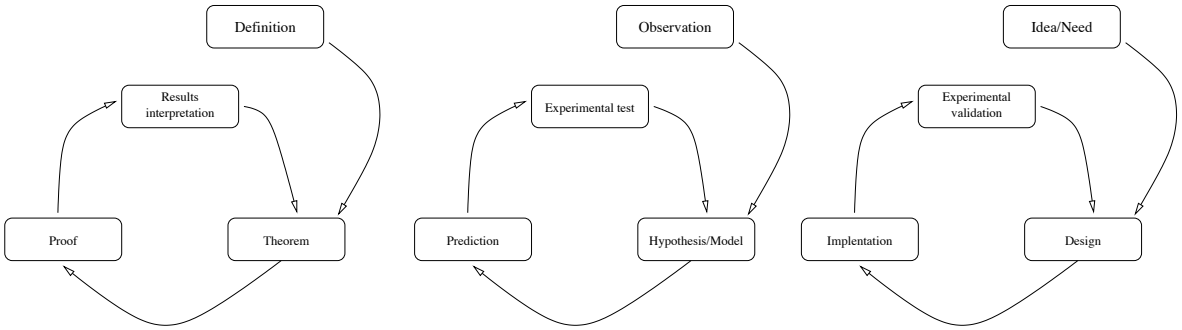


**Figure 1. The three paradigms of computer science: theory (left), modeling (center), design (right) [9, 11].**

experiments help in orienting research to promising directions.

### 2.3 Properties of experiments

Testing an implementation, validating a model or comparing two algorithms is routinely done in computer science and especially in research. However, having a highly meaningful experiment is difficult and requires meeting some prerequisites. In [1] we have proposed the four following properties[1] a *good experiment* should fulfill:

- **Reproducibility:** experimental conditions must be designed and described in such a way that they can be reproduced by other researchers and must give the same results with the same input. Such requirement is a real challenge for the computer science community as this description is often elusive and because, from one site to an other, environments are very different to each other.
- **Extensibility:** when targeting performance of an implementation or an algorithm, a scientific experiment report is of little interest if it simply describes the environment where the experiments have been conducted. Therefore, an experiment must target comparison with past and future results, extension with more or different processors, larger data set, or different architectures. Several dimensions must be taken into account such as scalability, portability, prediction or realism.
- **Applicability:** performance evaluation is only one aspect of experiment. The prediction of the behavior of an algorithm or a program in the real world is an other goal of experiments. However, the set of possible parameters and conditions is potentially infinite. A good experimental campaign must then be executed on realistic and representative set of entry parameters, input data, and usage. It must also allow a good calibration.
- **Revisability:** when experimental hypothesis are not met, a good experiment must help in identifying the reasons, may these reasons be caused by the modeling, the algorithm, the design, it implementation or the experimental environment. Methodologies must be developed to allow to explain errors and to find ways of improvements.

[1] we focus here to the case of distributed/parallel computing but this approach can easily be extended to the general case

## 3 Experimental methodologies

### 3.1 Properties of methodologies

In the case of distributed computing, the experimental validation of models, algorithms or programs is a difficult challenge. Indeed, in this case, the environments are very complex, at large scale, dynamic, and shared. Naives experiments on real platforms are often not reproducible, whereas, extensibility, applicability and revisability are hard to achieve. Such problems require to proceed by step as shown in figure 1 but also require tools that help to perform *good experiments*. In order to realize a good experiment we distinguish three methodologies, namely, simulation, emulation and in-situ. For each of these methodologies, correspond a class of tools: simulator, emulator and real-scale environments respectively. Whereas these tools are of different nature they need to share some common characteristics:

- **Control:** controlling experimental conditions is essential in order to know which part of the model or the implementation are evaluated. Moreover, the control allows testing and evaluating each part independently. Hence, in testing, several scenarios, experimental condition control helps in evaluating limits of models and proposed solutions. Controlling experimental condition requires being able to configure the environment which is a challenge by itself.
- **Reproducibility:** reproducibility is the base of the experimental protocol. The role of the experimental environment is to ensure the reproducibility. Hence an environment that ensures a good reproducibility must be considered better than an environment that ensures a lower one.
- **Realism:** Experimental conditions are always, in one way or an other, synthetic conditions. This means that they are an abstraction of the reality as they cannot take into account all the parameters. However, the level of abstraction depends on the chosen experimental environment: some are more realists than others. Therefore, when using an environment, it is important to know its level of realism in order to deduce what confidence can be put in the results. One can distinguishes three levels of realism:
  1. *Qualitative.* An experimental tool makes a qualitative comparison if, for instance, it says that Algorithm 1 is better than Algorithm 2 and in reality we have the same result
  2. *Quantitative.* An experimental tool makes a quantitative comparison if, for instance, it says that Algorithm 1 is 10 times better than Algorithm 2 and in reality we have the same result
  3. *Predictive.* An experimental tool makes a prediction if it says, for instance, that a program will last a given amount of time and in the reality we obtain the same result.

  It is important to note that predictive realism implies a quantitative one that implies a qualitative one.

### 3.2 Simulation

Simulators are the first kind of experimental tools. They focus on a given part of the environment and abstract the remaining of the system. Simulation allows performing highly reproducible experiments with a large set of platforms and experimental conditions. In a simulator only a model of the application is executed and not the application itself. Designing a simulator therefore requires a lot of modeling work and is therefore a very difficult scientific challenge.

### 3.3 Emulation

A second kind of experimental tools are emulators. An emulator targets to build a set of synthetic experimental conditions for executing a real application. Hence, contrary to the simulation, only the environment is modeled. We can distinguish two types of emulator. A first one is based on virtual machines that execute, in a confined way (a sandbox), the program. Each virtual machine can be executed on the same real machine and the network and CPU capabilities modified according to the targeted environment. This is for instance the approach taken by Microgrid [18]. A second approach consists in having the application executed directly on the hardware (without a virtual machine). In this case, the control of the environment (CPU or network speed) is done by degrading the performance. This is the approach taken by Wrekavoc [4] and will be discussed in the next section.

### 3.4 In-situ

In situ experiment is the last methodology. It is the one that offers the most realism as a real application is executed at a real scale using a real hardware. Such methodology is necessary because some complex behavior and interaction cannot always be easily captured and then simulated or emulated. This is the case, for instance for some operating system features (such as the process scheduling strategy, the paging algorithm, etc.); some hardware characteristics such as hyperthreading, cache management, multicore processors; runtime performance: two different implementation of the same standard (for instance MPI) can have different performances. However, using real machines may hinder the reproducibility as it is difficult to control the network traffic or the CPU usage (in case of shared machines). In order to tackle this problem, it is possible to design and build real scale environments dedicated to experiments. Among these environments we have: Das-3 [5], Grid'5000 [12, 2] or PlanetLab [15]. We will discuss the respective merits of these tools in the next section.

| Methodology | Simulation | Emulation (Virt. mach.) | Emulation (Degradation) | In-Situ (Homogeneous env.) |
|---|---|---|---|---|
| Real application | No | Yes | Yes | Yes |
| Abstraction | Very High | High | Low | No |
| Execution time | Speed-up | Slow-down | Same | Same |
| Proc. folding | Mandatory | Possible | No | No |
| Heterogeneity | Controllable | Controllable | Controllable | No |

**Table 1. Comparison of different experimental methodologies**

### 3.5 Comparison of these methodologies

We can compare the three above methodologies based on different criterion (see table 1):

- the ability to execute or not a real application. For emulation and in-situ a real application is executed, while for simulation only a model of these application is executed;
- the level of abstraction of the environment. The less abstraction, the more realism and the greater the confidence in the obtained results. This is related to the ability to execute the application and the underlying models used;
- the execution speed. Emulation tends to slowdown execution while simulation tends to speed it up (SimGRID as a speed-up factor of $10^6$ on certain applications on standard PC);
- the possibility to fold several CPUs on one. It allows to execute a parallel application on less (maybe one) processor(s) than in the reality. Simulators require only one CPU, while emulating or executing a parallel application is, most of time, done on several processors;
- the heterogeneity management. Is it possible to have heterogeneity? If yes, is this heterogeneity controllable? In-situ provides a fix hardware setting with little heterogeneity while emulator and even more simulator provide a way to manage and control a very high heterogeneity.

What clearly shows table 1 is that each methodology has advantages and drawbacks. This means that depending of the experimental goal it is important to carefully choose the corresponding methodology and the tool.

## 4 Example of experimental tools

Several tools have been developed in the recent years to implement the above methodologies. We describe (and eventually compare some of them) here in the context of grid and distributed computing.

### 4.1 Tools for simulation

There exists tremendous number of simulators that model CPU or network behavior (see [16] for a taxonomy). However in the context of grid computing there are two very lively projects respectively GridSim[2] [3] and SimGRID[3] [13]. They both share the same goal: simulating a parallel application on a grid/distributed environment. It allows to create a distributed environment (resources connected by network links) and simulate tasks and communications onto these resources. In terms of functionality these two simulators propose different approaches. In terms of realism, the network GridSim model uses a packet fragmentation method where each packet is sent with the corresponding delay. This means that the simulation time is proportional to the size of the messages as for packet-level simulator such as the well-known GTNets[4]. However, contrary to such low level simulator, GridSim does not provide a fine modeling (it is not possible to set the TCP congestion window size for instance) which hinder the quality of the simulation. On the other hand, SimGRID provides a fast method to simulate the network, where the simulation time is propositional to the number of events (start or end of a network stream) while providing at least the same realism than GridSIM. If a greater realism is required, it is possible to use GTNets within SimGRID at the cost of a higher simulation time but without changing the code.

### 4.2 Emulators

When simulation does not provide enough accuracy, emulation (by executing the real applica-

[2] http://www.buyya.com/gridsim
[3] http://simgrid.gforge.inria.fr
[4] http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS

tion) can help in having more realism.

The first grid emulator was Microgrid [18]. It allows to emulate the parallel execution of grid application using some (possibly one) processor(s). Each, emulated resource and executes, in a confined way, part of the applications. Some system calls (such as IO) are intercepted by the Microgrid library and emulated according to the modeled environment. Experimental validation show that Microgrid is able to emulate tens of thousands resources. The main problem with Microgrid is that the project is not evolving anymore and it does not compile on recent systems.

We have designed a tool called Wrekavoc[5] [4] to tackle the problem of emulating a heterogeneous environment. Wrekavoc addresses the problem of controlling the heterogeneity of a cluster. The objective is to have a configurable environment that allows for reproducible experiments on large set of configurations using real applications with no simulation of the code. Given an homogeneous cluster Wrekavoc degrades the performance of nodes and network links independently in order to build a new heterogeneous cluster. Then, any application can be run on this new cluster without modifications. However, contrary to Microgrid, Wrekavoc needs one real CPU per emulated CPU.

We have implemented several methods for degrading CPU performance. The first approach consists in managing the frequency of the CPU through the kernel CPU-Freq interface. We propose two other solutions in case CPU-Freq is not available. One is based on CPU burning. A program that runs under real-time scheduling policy burns a constant portion of the CPU, whatever the number of processes currently running. The other is based on user-level process scheduling called CPU-lim. A CPU limiter is a program that supervises processes of a given user. Using the `/proc` pseudo-filesystem, it suspends the processes when they have used more than the required fraction of the CPU.

Limiting latency and bandwidth is done using *tc* (traffic controller) based on *Iproute2* a program that allows advanced IP routing. With these tools it is possible to control both incoming and outgoing traffic. Furthermore, the latest versions (above 2.6.8.1) allow to control the latency of the network interface.

Wrekavoc is able to limit the amount of memory available by the processes thanks to the use of `mlock` and `munlock` system call that pins some memory pages into the physical memory.

The configuration of an homogeneous cluster is made through the notion of islet. An islet is a set of nodes that share similar limitation. Two islets can be linked together by a virtual network which can also be limited or by an intermediate islet (in this case packets are routed from islet to islet). The notion of islets then provides a simple way to describe a virtual topology that emulates a given heterogeneous environment (possibly a grid) using an homogeneous cluster.

To validate Wrekavoc, we have compared the results of our tool on a cluster composed of homogeneous nodes with the results obtained on a real heterogeneous cluster composed by twelve workstations (from Pentium II to Pentium IV). We used several software like Povray, parXXL, some matrix multiplications algorithm and others parallel software to make some comparisons. Results show that Wrekavoc is very realistic both quantitatively and qualitatively. For instance, concerning parallel image rendering we have used an MPI program that decomposes a Povray image model and sends each part of the model to a processor. It then reassembles the parts of the image when the processors send them back to it. We have run this program on our heterogeneous cluster and on the same emulated cluster using Wrekavoc. Results show that the amount of work done by each processor is the same in both cases. A movie showing this realism is available at http://wrekavoc.gforge.inria.fr/html/demo-wrekavoc.avi and a screenshot of this movie is shown in Fig. 2.

### 4.3 Real scale experimental tools

There are very few environments that are designed for performing experiments at real-scale in the domain of grid and distributed computing. Among these tools we can cite Grid'5000 [12, 2], Das-3 [5] or PlanetLab [15].

The purpose of Grid'5000 is to serve as an experimental testbed for research in grid computing and large-scale systems. In addition to theory, simulators and emulators, there is a strong need for large scale testbeds where real life experimental conditions hold. Grid'5000 aims at building a highly reconfigurable, controllable and monitorable experimental Grid platform gathering nine sites geographically distributed in France. Each site hosts from one to 3 clusters that can be reserved to conduct large-scale experiments. Each site is connected to the other using a dedicated network with link of either 1 or 10 Gbit/s bandwidth (see Figure 3 for the topology). Grid'5000 features a total of more than three thousands CPUs and four thousands cores. Each node of

[5] http://wrekavoc.gforge.inria.fr

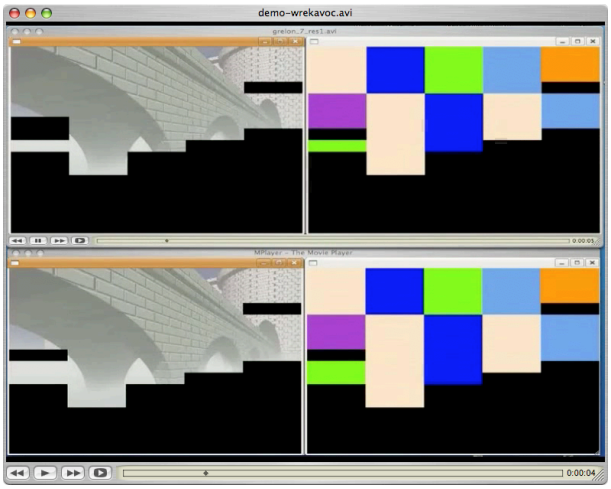


**Figure 2. Screenshot of the movie showing the parallel rendering of an image using Povray. The lower part shows the rendering using a real heterogeneous cluster. The upper part shows the same rendering using Wrekavoc that emulates the heterogeneous cluster. On the left of each part is the rendered image. On the right is the processor repartition. The full movie is available at** `http://wrekavoc.gforge.inria.fr/html/demo-wrekavoc.avi`

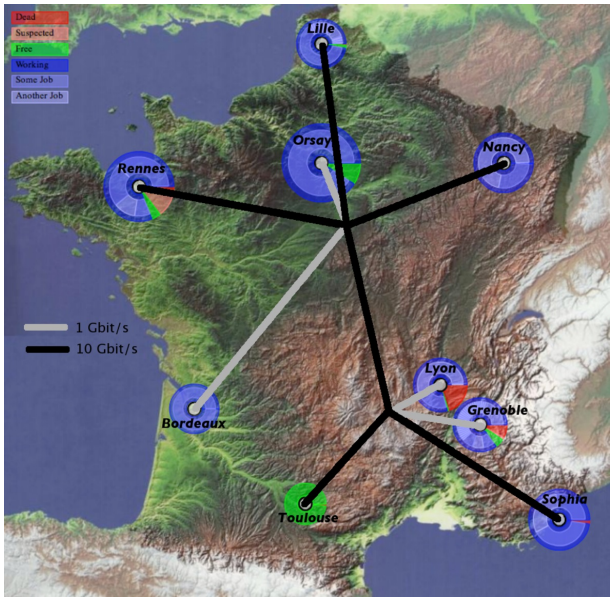


**Figure 3. Grid'5000 sites interconnection**

each cluster is configurable using the Kadeploy[6] tool. Kadeploy allows to set-up, custom and boot the own system of the experimentalist on all the reserved nodes of a given cluster. The user can then install desired software, configure and parametrized the system, and save and retrieve this environment when needed. Hence for each experiment the user can reconfigure the whole software stack from the network protocol to the application.

The Dutch project DAS-3 targets the same goal as Grid'5000 but at a smaller scale (Four clusters and 270 dual CPUs node). Contrary to Grid'5000 where users can install and configure their own system, DAS-3 does not provide such configurable environment. However, Das-3 has been designed to allow the reconfiguration of the optical network that interconnects the 4 sites. An on-going project is two connect these two testbeds at the European level.

Lastly, PlanetLab is a planet-wide testbed. The goal of PlanetLab is to enable large-scale experiments under real world conditions. It has fewer nodes than Grid'5000 (825) but a larger number of sites (406). Hence, projects undergone on PlanetLab are more oriented to peer-to-peer, network or distributed algorithms while on Grid'5000 the focus is more on "*cluster of clusters*" issues. Each PlanetLab node embeds a software package in order to support distributed virtualization. Thanks to this virtualization, different experiments, using different but possibly overlapping set of nodes (called slices) can be performed concurrently.

[6] http://kadeploy.imag.fr/

## 5 Conclusion

Computer science is an experimental science. This is particularly true for grid and distributed computing where the environments are very hard to model analytically. A solution to this problem is to experimentally test, validate or compare the proposed solutions. However, defining and conducting a *good experiment* is a difficult task especially in the context of large-scale systems. In this paper, we have presented the role and the properties of experiments in this context. We have then described three complementary methodologies (namely simulation, emulation and real-scale) that allows to perform *good experiments*. Lastly we have described and compared several tools that implement these methodologies. Such tools allow researchers to perform experiments in order to test, validate or compare their proposed solu-

tions.

## 6 Acknowledgement

We would like to thank Martin Quinson for fruitful discussions on this subject and Olivier Dubuisson for his work on the validation of Wrekavock.

## References

[1] Algorille Team, Algorithms for the Grid. INRIA Research proposal, July 2006. Available at http://www.loria.fr/equipes/algorille/algorille2.pdf.

[2] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, November 2006.

[3] Rajkumar Buyya and M. Manzur Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *CoRR*, cs.DC/0203019, 2002.

[4] L.-C. Canon and E. Jeannot. Wrekavoc a Tool for Emulating Heterogeneity. In *15th IEEE Heterogeneous Computing Workshop (HCW'06)*, Island of Rhodes, Greece, April 2006.

[5] The DAS-3 project: http://www.starplane.org/das3/.

[6] Peter J. Denning. ACM President's Letter: What is experimental computer science? *Commun. ACM*, 23(10):543–544, 1980.

[7] Peter J. Denning. ACM president's letter: performance analysis: experimental computer science as its best. *Commun. ACM*, 24(11):725–727, 1981.

[8] Peter J. Denning. Is computer science science? *Commun. ACM*, 48(4):27–31, 2005.

[9] Peter J. Denning, D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, 1989.

[10] D. D.Johnson. A theoretician's guide to the experimental analysis of algorithms, 2001. AT&T Labs Research. Available from http://www.research.att.com/~dsj/papers/experguide.ps.

[11] Dror G. Feitelson. Experimental Computer Science: The Need for a Cultural Change. Internet version: http://www.cs.huji.ac.il/~feit/papers/exp05.pdf, December 2006.

[12] The Grid 5000 project: http://www.grid5000.org/.

[13] Arnaud Legrand, Loris Marchal, and Henri Casanova. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *CCGRID*, pages 138–145, 2003.

[14] Paul Luckowicz, Walter F. Tichy, Ernst A. Heinz, and Lutz Prechelet. Experimental evaluation in computer science: a quantitative study. Technical Report iratr-1994-17, University of Karlshrue, Germany, 1994.

[15] Planet lab: http://www.planet-lab.org/.

[16] Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Softw., Pract. Exper.*, 34(7):653–673, 2004.

[17] Walter F. Tichy. Should Computer Scientists Experiment More? *Computer*, 31(5):32–40, 1998.

[18] Huaxia Xia, Holly Dail, Henri Casanova, and Andrew A. Chien. The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments. In *CLADE*, page 52, 2004.

[19] M.V. Zelkowitz and D.R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, May 1998.