Parallel Processing Letters

Experimental Methodologies for Large-Scale Systems: a Survey

Jens Gustedt
AlGorille team and LORIA
INRIA Nancy Grand-Est, France

Emmanuel Jeannot
AlGorille team and LORIA
INRIA Nancy Grand-Est, France

Martin Quinson
AlGorille team and LORIA
Nancy University, France

Received December 2008
Revised March 2009
Communicated by J. Dongarra and B. Tourancheau

ABSTRACT

The increasing complexity of available infrastructures with specific features (caches, hyperthreading, dual core, etc.) or with complex architectures (hierarchical, parallel, distributed, etc.) makes it extremely difficult to build analytical models that allow for a satisfying prediction. Hence, it raises the question on how to validate algorithms if a realistic analytic analysis is not possible any longer. As for some many other sciences, the one answer is experimental validation. Nevertheless, experimentation in Computer Science is a difficult subject that today still opens more questions than it solves: What may an experiment validate? What is a "good experiment"? 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", mainly in the context of parallel and distributed computing. More precisely we will focus on four main experimental methodologies, namely in-situ (real-scale) experiments (with an emphasis on PlanetLab and Grid'5000), Emulation (with an emphasis on Wrekavoc) benchmarking 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.

Keywords: Experimental computer science; experimental methodologies; in-situ; emulation; benchmarking; simulation

1. Introduction

Being a relatively young science, the scope and techniques of *Computer Science* are not commonly agreed upon among computer scientists themselves, and probably even less when it comes to non-specialist. In large parts, this is probably due to historical reasons, since the social roots of early computer scientists have at least been double; on one side there is a strong historical link to mathematics and logic, and on the other hand one to engineering sciences. This difference in view already expresses in the term that is applied for it: the strong reference to *computers* (or sometimes computing) in the English term is completely absent in other languages that prefer to coin the science *Informatics*, and thus stress more on the aspect of 'science of information'.

But in fact modern Computer Science is much more than 'just' an off-spring of mathematics and engineering but now finds itself as a proper discipline with intersections to biology, cognitive science, computer engineering, economics, linguistics, mathematics, physics, philosophy, social science, and probably others. When we think of these intersections to the other sciences, we often have the tendency to emphasize upon the modeling aspect (*e.g* neural networks as being inspired by biology) or the objects under investigation (computers and/or information), we often neglect the methodological aspects that we share with all of them. The aim of the present paper is to describe such an overlap in methodology, namely the aspect of *Computer Science being an experimental science*.

Suppose first as a working hypothesis that Computer Science is the science of information. As stated by Peter J. Denning et al. in [1], "*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 analytic work using mathematics as a tool. Such an approach has been proven very efficient to provide a stable theoretical basis of Computer Science but is much limited when the studied object becomes too complex. In the case where the objects (hardware, program, algorithms, etc.) are hard to understand analytically or formal specifications are simply not available, a second approach, based on experiments (see [2]) allows to gather and classify knowledge through observations. Such observations can be done either *in nature* or under *laboratory conditions*. The first is present in activities such as collecting traces of network connections, much similar the way biologist gather information about biological species. This then leads to a *taxonomy* or classification of the objects under investigation, such as the distinction of network flows in *mice and elephants*, see [3].

By establishing such a taxonomy we observe that modern systems such as network, computers, programs, algorithms, protocols and services 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 large-scale systems because they are composed of a stack of these objects.

If the role of experiments becomes more and more important in computer science and especially in distributed computing, it is noticeable that the experimental culture of computer scientists is not comparable to others in fields that consensually understand themselves as experimental such as biology or physics. For instance, a study led by Luckovicz et al. [4] in the early 90's[a] shows that among published Computer Science articles in ACM journals, between 40% and 50% of those that require an experimental validation had none. The same study observes that this ratio is only 15% for journals in *optical engineering* in physics. A statistic study published in 1998 [6] 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 seems to decrease in 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 the natural sciences, even if it is improving (at least quantitatively).

Besides the historical reasons that we mentioned above, there are other reasons for this insufficiency: a chronic lack of financing for experimenter positions, missing disposability of dedicated experimental environments, and a general difficulty of esteem of work-intensive experimental results in scientific careers. But a general observation also shows that we, as a young science, still lack methodologies and tools to conduct scientific experiments.

Thus, the goal of this article is to discuss and study the importance of experiments in Computer Science. We focus on large-scale and distributed systems 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 will be done in Section 4. Finally, we will conclude in Section 5.

2. Role of experiment in Computer Science

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

2.1. *Related work*

The question of whether or not Computer Science is a science is brightly and affirmatively answered in [7]. The point is that, even if there are some disagreements between computer scientists on whether

[a]Also taken up again by Tichy [5].

Computer Science is also technology, art, engineering, hacking, etc., there is no doubt that a large part of the discipline is in fact science, and this should not be seen as a contradiction but as a complement to other aspects. In [2] and [8], Denning discusses on the importance of experiment in Computer Science. To give an example, he highlights that the LRU paging algorithm has been proven better then the FIFO strategy experimentally.[b]

As stated in the introduction the lack of experiments in computer has been emphasized in [4, 5]. In [9], D. G. Feitelson shows that experiments 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 the above issues in detail.

2.2. *Role of experiments*

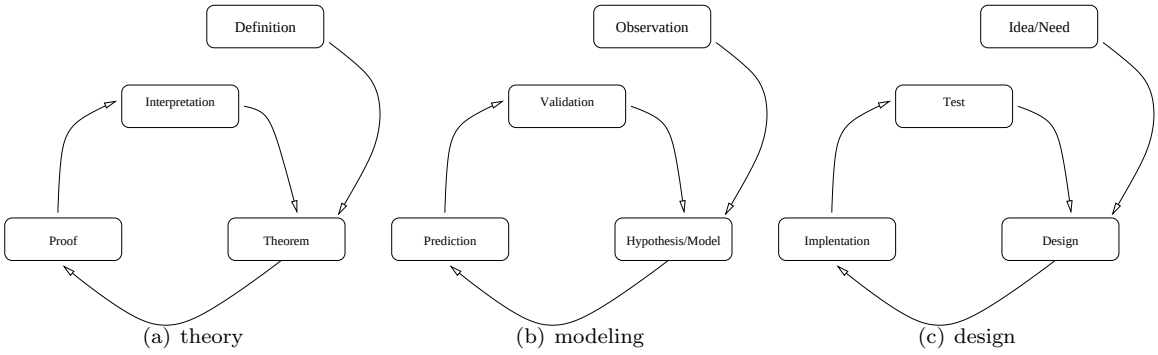

Fig. 1. The three paradigms of computer science [1, 9].

In [1], Denning et al. present three paradigm of the discipline of computing: theory, 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 we seek to construct (*i.e.* a theory, a model or a program). These three feedback loops are also used by Feitelson in [9] and are shown in Fig. 1. We first want to emphasis that these three methodologies are complementary. Let us take the example of algorithm design. The investigation of an algorithm must use a model of the reality that has to be experimentally validated. With the use of a formal description it is possible to demonstrate some property of this algorithm (complexity, approximation ratio, etc.). Moreover, an implementation of this algorithm will allow to benchmark real cases for efficiency.

As shown in this example and in Fig. 1, one can remark that the modeling (Fig. 1(b)) and the design paradigms (Fig. 1(c)) are driven in large parts by experiments and tests. The first paradigm (Fig. 1(a)) also knows about experiments for verifying or falsifying conjectures (*e.g* big efforts are undertake to find prime numbers of a special type) or for generating objects with desired properties (*e.g* specialized hash functions). But these are deployed much less frequently than for the two others paradigms, and will not play a major role in this paper. According to these three paradigms we observe three different principal types of experiments:

- Explore the combinatorial states of a theory to falsify or generate conjectures.
- Validate a model in comparing its prediction with experimental results.
- Test the design quantitatively, *i.e.* by measuring the performance of design under normal conditions.

[b]LRU = Last Recently Used; FIFO = First In First Out

However, it is important to understand that different validations may occur simultaneously. 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 [5], Tichy presents the main advantages of experiment: in testing hypothesis, algorithms or programs, experiments 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 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, conducting a meaningful experiment is difficult and requires meeting some prerequisites. In [11], we have proposed the four following properties[c] a *good experiment* (in the sense of *good practice* for an experiment) should fulfill:

- **Reproducible:** 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. However, this aspect is more and more addressed in the Computer Science community. For instance, in the ACM SIGMOD 2008 conference[d] on databases, it was required that the authors send their experimental input and program along with the submitted paper to allow the program committee to redo the experiments. Such approach could be extended to other conferences at the cost of a longer reviewing process and provided that the authors are able to package the software and the experimental conditions in a clean and simple way.
- **Extensible:** 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. In 2000 Cappello et al. [12] showed that the hybrid programming model MPI+OpenMP was less performing that the plain MPI programming model using an IBM SP machine of 32 nodes. This results was valid with modern clusters until the appearance of multicore processors for which (with some change in the data structure), the hybrid model seems to be better.
- **Applicable:** performance measurement is only one aspect of experiments. 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. For instance in [13] Tsafrir et al. show that real workloads contains anomalies (called flurries) that are not representative and hence have to be pruned for a good modeling.
- **Revisable:** 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, its implementation or the experimental environment. Methodologies must be developed to allow to explain errors and to find ways of improvements. A nice example of model invalidation was done by Paxson et al in [14]. They showed that the well used Poisson model of wide area TCP arrival process was not matched in any cases and that other models (long-tail distributions) have to be used.

[c] We focus here to the case of distributed/parallel computing but this approach can easily be extended to the general case.
[d] http://www.sigmod08.org/sigmod_research.shtml

2.4. *The case of large-scale and distributed systems*

Large-scale and distributed systems such as grids or clouds are good examples where the experimental approach is necessary. Indeed, these systems are built using computers that have very advanced features (caches, hyperthreading, 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 middleware 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. Finally, failures, shared usages, etc. make the behavior hard to predict. This is why we think that such systems are a good example of the necessity of the experimental approach. Hence, we will focus on these environments in the following sections.

3. Experimental methodologies

3.1. *Different types of methodologies*

As stated above, the experimental validation of models, algorithms or programs is a difficult task. Naive 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 Fig. 1 but also require tools that help to perform *good experiments*.

Table 1. Our classification of the four experimental methodologies (most examples are given in the context of large-scale systems).

		Environment	
		Real	*Model*
Application	*Real*	**In-situ** (Grid'5000, DAS3, PlanetLab, GINI, ...)	**Emulation** (Microgrid, Wrekavock, V-Grid, Dummynet, TC, ...)
	Model	**Benchmarking** (SPEC, Linpack, NAS, IOzone, ...)	**Simulation** (SimGRID, GRIDSim, NS2, PeerSim, P2PSim, DiskSim, ...)

In order to test or validate a solution, one need to execute a real (or a model of an) application on a real (or a model of an) environment. This leads to four classes of methodologies as shown in Table 1: *in-situ* where we execute a real application (program, set of services, communications, etc.) on real a environment (set of machines, OS, middleware, etc.), *emulation* where we execute a real application on a model of the environment, *benchmarking* where we execute a model of an application on a real environment and lastly *simulation* where we execute a model of an application on a model of the environment. We strongly think that such classification is very general and applicable to other domains that large-scale systems. However as stated above the examples will be given for this context.

3.1.1. *In-situ*

In situ experiments offer the least abstraction 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, especially 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 [15], Grid'5000 [16, 17] or Planet-Lab [18]. We will discuss the respective merits of these frameworks in Section 4.1.

3.1.2. *Emulation*

An emulator targets to build a set of synthetic experimental conditions for executing a real application. Hence, contrary to the in-situ approach, the environment is modeled. We can distinguish two types of emulators. A first one is based on virtual machines that execute the program in a confined way (a sandbox). If plain virtual machine systems such as Xen [19] or QEMU [20] are not real emulators (they are not designed for that purpose), virtualization is a technology that can be used as the basis for emulation as it allows several guests to be executed on the same physical resource. This is for instance the approach taken by Microgrid [21]. 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 [22]. Both approaches will be discussed in Section 4.2.

3.1.3. *Benchmarking*

Benchmarking consists in executing a model of an application (also called a synthetic/generic application) on a real environment. In general the goal is to quantitatively instrument these environments (being the hardware, the operating system, the middleware, etc.). Well-known examples of benchmarks are the HPL Linpack[e] (when used for the TOP 500 ranking[f]), the NAS parallel benchmarks [23], or the Montage workflow [24] (when used to measure characteristics of workload schedulers).

3.1.4. *Simulation*

Simulators focus on a given part of the environment and abstract from the rest of the system. Simulation allows to perform 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. Well known simulators in Computer Science include SimOS [25] for Computer Architecture, NS2[g] and GTNetS [26] for Networking, as well as GridSim [27] and SimGrid [28] for Grid Computing.

3.2. *Properties of methodologies*

To each of the methodologies described in the previous section corresponds a class of tools: real-scale environments, emulators, benchmarks and simulators. Whereas these tools are of different nature, they need to share some common characteristics. We distinguish two classes of properties. *General properties* concern properties that are considered for any experimental tool in Computer Science (such as reproducibility). *Large-scale system properties* are related to experimental tools suited for grid or clouds and other large-scale/parallel environments (such as the heterogeneity management). However it is important to understand that some of the properties are conflicting. For instance it is very difficult to have a tool that simultaneously realizes good scalability, good realism and fast execution speed. Therefore, each tool has to deal with a trade-off between these properties.

3.2.1. *General properties*

- **Control:** controlling experimental conditions is essential in order to know which properties of the model or the implementation are measured. Moreover, the control allows testing and measuring each part independently. Hence, in testing, several scenarios, experimental condition control helps

[e] http://www.netlib.org/benchmark/hpl
[f] http://www.top500.org/
[g] http://nsnam.isi.edu/nsnam/

in evaluating limits of models and proposed solutions. Controlling experimental condition requires the configuration of 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 this feature. Hence an environment that ensures good reproducibility must be considered better than an environment that ensures less.
- **Abstraction:** 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, it is important to know its level of abstraction in order to deduce what confidence can be put into the results. We distinguish three levels of confidence:
  (1) *Qualitative.* An experimental tool makes a qualitative comparison if, for instance, it correctly states that Algorithm 1 is better than Algorithm 2.
  (2) *Quantitative.* An experimental tool makes a quantitative comparison if, for instance, it correctly states that Algorithm 1 is 10 times '*better*' than Algorithm 2.
  (3) *Predictive.* An experimental tool makes a prediction if, for instance, it states the runtime of a program correctly.

  It is important to note that predictive implies quantitative which implies qualitative.

3.2.2. *Large-scale system properties*

- **Scale:** grid and peer-to-peer environments are large-scale systems. In order to test solutions for such systems experimental tools need to scale to a large number of entities (network link, processors, users, etc.) in order to demonstrate how such solution behave when the number of components increases to what is considered a usual case.
- **Execution speed** concerns the speed-up factor between experiment and a real execution. For instance, emulation tends to slowdown execution while simulation tends to speed it up[h]. This is an important property since faster execution allows for more tests and thus more comprehensive testing campaigns.
- **Processor folding** is 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. Most simulators require only one CPU, while emulating or executing a parallel application is, most of time, done on several processors.
- **Heterogeneity management of the platform:** 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.

4. Example of experimental tools for large-scale systems

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 large-scale and distributed computing.

4.1. *In-situ experimental tools*

An in-situ experiment consists in executing a real application on a real environment (see Sec 3.1.1). The goal is to have a setting that is general enough such that conclusions found with such environment are extensible to other cases.

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 [16, 17], Das-3 [15] (both targeting grids) or PlanetLab [18] (targeting peer-to-peer environments, distributed storage, etc.).

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

[h]While most of the time the execution speed is speed-up by simulation, there exists notable exceptions such as CPU simulator and some network simulator. It mostly depends on the level of abstraction of the models used in the simulator

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 Fig. 2 for the topology. Grid'5000 features a total of more than three thousands CPUs and four thousands cores. Each node of each cluster is configurable using the Kadeploy[i] 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 parametrize 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.

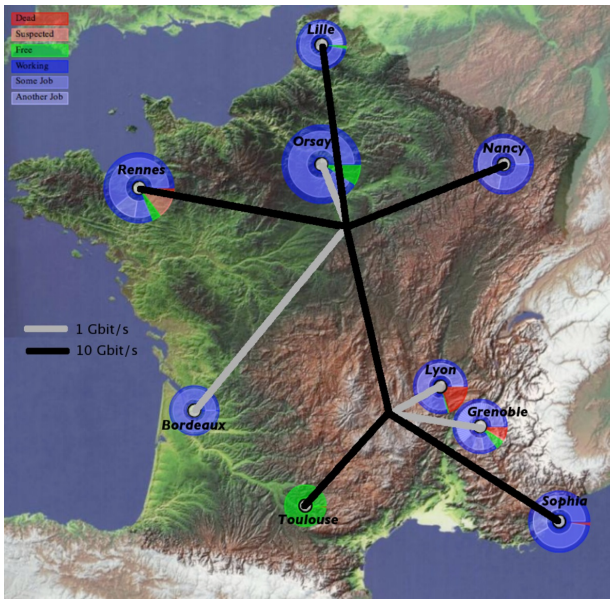

Fig. 2. Grid'5000 sites interconnection

The Dutch project DAS-3 targets the same goal as Grid'5000 but at a smaller scale (Four clusters and 270 dual CPU nodes). 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. Based on PlanetLab, Everlab [29] is a testbed that supports both experimentation and production. It works through a private PlanetLab network across all clusters of this testbed.

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

Table 2. Characteristics of some in-situ experimental tools.

Tools	PlanetLab	DAS-3	Grid'5000
Control	Very low	Network topology	Software stack
Reproducibility	Low	Average	Average
Abstraction	Very low	Very low	Very low
Scale	1000s	100s	1000s
Execution time	Same	Same	Same
Proc. folding	Possible	No	No
Heterogeneity	Fixed	Fixed	Fixed

Table 2 compares the properties of the three above in-situ experimental tools. Due to the fact that, in Grid'5000, the software stack is entirely configurable and tunable, the control of the experimental condition is much better than in PlanetLab or DAS-3 (though it is not perfect due to cross-traffic on the network).

4.2. *Emulators*

In emulation, contrary to in-situ experiments, only a model of the environment is used to execute a real application (see Sec 3.1.2). Building an emulator therefore requires to build a model of the environment that should be realistic enough to minimize abstraction but simple enough to be tractable. An other aspect is that, in emulation, a given application is run (almost) unmodified and therefore requires little engineering effort.

The RAMP (Research Accelerator for Multiple Processors) project [30] aims at emulating low level characteristics of an architecture (cores, cache, memory bus, etc.) using FPGA. The goal is to build a very precise environment with as little abstraction as possible. The downside, of such tool is that the scale is very low (in the order of ten CPUs). Moreover, as far as we know, there is nothing concerning the network in RAMP.

ModelNet [31] is a tool that is principally designed to emulate the network components. It works at real-scale under a real hardware. Network packets are routed through core ModelNet nodes which cooperate to limit the traffic to profile of a target network topology in terms of the bandwidth, of congestion constraints, of latency, and of packet loss. Experiments show that compared to real applications the observed behavior is similar. ModelNet has been used with experiments using more than 100 machines. Emulab [32] shares similar goals with ModelNet and provides network emulation.

There exists other network emulators such as TC [33] or dummynet [34]. Both tools work similarly by intercepting packets and delay, slowdown or delete them according to traffic shaping rules given by the experimentalist.

Microgrid [21] allows to emulate the parallel execution of distributed application using some (possibly one) processor(s). It emulates both the CPU and the network performance. Each, emulated resource executes, in a confined way, part of the applications. Some system calls (`gethostbyname`, `bind`, `send`, `receive`) are intercepted by the Microgrid library and emulated according to the modeled environment. This technique is invasive and limited to applications that are integrated into Globus. Experimental validation show that Microgrid is able to emulate tens of thousands resources. However, the measured resource utilization seems to be relatively inaccurate, in particular due to its round-robin scheduler. Moreover and unfortunately, Microgrid is not supported anymore and does not work with the recent compiler versions such as `gcc 4`.

We have designed a tool called Wrekavoc[j] [22] 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

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

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. The network limitation is based on TC and the CPU degradation is based on user-level process scheduling (processes are suspended when they have used more than the required fraction of the CPU). Validation of Wrekavoc has been done by comparing the execution of a real application on a real heterogeneous cluster and cluster emulated with Wrekavoc. Results show that Wrekavoc is able to have a very realistic behavior, once it is carefully calibrated.

Some emulators are also integrated within a development environment. This is the case of SPLAY-Ground that features an emulation of a large-scale distributed system of 200 nodes for applications written using the SPLAY environment[k].

Table 3. Characteristics of some emulators.

Tools	RAMP	ModelNet	MicroGRID	Wrekavoc
Control	High	High	High	High
Reproducibility	Perfect	High	High	High
Abstraction	Very low	Very low	High	Low
Scale	10s	100s	100s	100s
Execution time	Slow down	Slow down	Slow down	Same
Proc. folding	No	No	Yes	No
Heterogeneity	Controllable	Controllable	Controllable	Controllable

Table 3 compares the properties of emulators. Due to the fact that the RAMP system uses FPGA to emulate low level hardware feature the abstraction and the reproducibility is expected to be better than for the three other tools. This is done at the cost of a much smaller scale. As the code is directly executed on the real hardware, Wrekavoc does not slowdown the execution time.

4.3. *Benchmarks*

A benchmark consists in executing a model of an application (a synthetic/generic application) on a real environment (see Sec 3.1.3). The application model should be generic enough to capture and measure some important characteristics of the target environment (*e.g.* CPU speed, network characteristics, IO performance, etc.). However, it is not a real application in the sense that it has to be synthetic and nobody really cares for the results (think for instance of the number of *random matrices* that have been factorized or multiplied to bench an environment).

A good example of a grid benchmark is the *montage* workflow. This workflow is a directed acyclic graph that models a parallel image processing application for astronomy. Each level of the workflow represent a phase in the processing, and the width represent the number of input images. This workflow has been used in several papers to measure characteristics of a middleware, a workflow engine or a scheduler [24, 35].

Another well-known set of benchmarks are the NAS parallel benchmarks [23]. It is a set of program designed to measure characteristics of parallel environment. There are 5 kernels and 3 synthetic applications (coming from computational fluid dynamics). It has been used for years to measure and compare parallel solutions (algorithms, machines, systems). Interestingly, a given benchmark is able to capture a part of the environment while another benchmark captures a different part: for instance, *Embarrassingly Parallel* (EP) is suited for capturing the accumulated floating point performance of a system while *Integer Sort* (IS) measures both integer computation speed and communication performance.

The situation of the Linpack program is more blurred. In one sense, it is a real software library used to perform linear algebra computation, but it is also very well known in the context of the TOP'500 list[l] that intends to classify the 500 most powerful parallel machines in the world. Each such machine is ranked based on the Linpack performance (measured in GFlops) obtained on it. In this case Linpack is

[k]http://www.splay-project.org/
[l]http://www.top500.org/

used as a benchmark and not as a real application for which the output is of some importance. However, ranking machines using the Linpack benchmarks poses several well-known problems such that the fact it is mainly a CPU intensive benchmarks as it is possible to scale-up the matrix size to coarsen the computational grain. Moreover, not every company can stop its machine to run such a benchmark and enter the TOP500 list (think of the Google clusters for instance).

Table 4. Characteristics of some benchmarks.

Tools	Montage workflow	NAS	Linpack
Control	Low	Average	None
Reproducibility	High	High	High
Abstraction	High	Average	High
Scale	10s	1000s	10^6s
Execution time	Same	Same	Same
Proc. folding	No	No	No
Heterogeneity	Slightly Controllable	Fixed	Fixed

Table 4 compares the properties of benchmarks. The control of the experimental conditions is best done with the NAS benchmarks as there is several kernel each measuring different parts of the environment. In the contrary, the fixed nature of Linpack does not offer any control of the conditions. For the same reason the abstraction is lower with the NAS benchmark while the montage workflow and the Linpack application cover only one use-case. The heterogeneity is slightly controllable with the Montage workflow as the width and the graph can be enlarged and each task composing the graph is different.

4.4. *Simulators*

Simulation is a classical approach in science, for example often used by physicists and biologists in their experiments. It is also widely used in Computer Science. In simulation, a model of an application is executed on a model of an environment (see Sec 3.1.4). This is why, in most of the cases the reproducibility is very high.

There exists tremendous number of simulators that model CPU at the hardware level for Computer Architecture studies (see [36] for a taxonomy). In Networking, the most common tool is certainly *The Network Simulator* (NS2)[m], which grounds most of the research literature of the field. Some competitors exist, such as SSFNet [37], which constitutes a standard API benefiting of several implementations or GTNetS [26], which improves the scalability issues of NS2.

The situation is a bit different in simulation of Large-Scale Distributed Systems, since most authors prefer to develop their own simulator instead of relying on an existing and widely accepted methodology. In fact, [38] reports that out of 141 papers about P2P and based on simulation, 30% use a custom simulation tool while half of them do not even report which simulation tool was used! Most of these tools only intend to be used by their own developers, and even when they are released to the public they target a very specific community.

The emerging tools used in Peer-to-Peer community encompass PlanetSim [39] and PeerSim [40]. Their main goal is scalability (from hundreds of thousands of nodes, up to millions of nodes), at the price of a very high level of abstraction. Computation times are simply ignored while communication time is supposed to take a fixed predetermined amount of time (these tools are thus said to be discrete time simulators). In PeerSim, the abstraction is even pushed further since the simulated application must be expressed as a simple state machine, allowing the runtime to use a very efficient representations for this. This methodology makes it possible to simulate several millions of nodes on a single host.

Recent Grid Computing articles are usually grounded by one of three well known simulators. OptorSim [41] emphasis on data storage and transfer, explaining that it is very commonly used in Data Grid research. It does not take computation time into account, making its use difficult for generic Grid

[m] http://nsnam.isi.edu/nsnam/

Computing research. Both GridSim[n] [27] and SimGrid[o] [28] aim at simulating a parallel application on a distributed grid environment. GridSim is mainly focused on Grid Economy studies while SimGrid's authors claim a larger application field to any application distributed at large scale. The scalability comparison conducted in [42] shows the clear advantage of SimGrid over GridSim to that extend[p]. This difference mainly comes from the underlying models used to represent the network. GridSim 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. But since it does not take TCP congestion control mechanisms, its realism is still to be assessed. On the other hand, SimGrid provides a fast method to simulate the network, where the simulation time is proportional to the number of events (start or end of a network stream). The validity of this approach was shown in [43]. 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.

Table 5. Characteristics of some simulators.

Tools	NS2	GTNetS	OptorSim	SimGrid	GridSim	PlanetSim	PeerSim
Control	Very High	Very High	Very High	Very High	Very High	Very High	Very High
Reproducibility	Perfect	Perfect	Perfect	Perfect	Perfect	Perfect	Perfect
Abstraction	Average	Average	High	High	High	Very High	Very High
Scale	100s	1000s	100s	$10^4 s$	1000s	$10^5 s$	$10^6 s$
Execution time	Equivalent	Equivalent	Faster	Faster	Faster	Faster	Faster
Proc. folding	Often	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
Heterogeneity	Controllable	Controllable	Controllable	Controllable	Controllable	Controllable	Controllable

Table 5 compares different simulators for Networking and Large-Scale distributed systems. Any such tool provides a very high control of the experimental conditions (only limited by tool scalability), and a perfect reproducibility by design. Likewise, the heterogeneity of the experimental environment can be perfectly controlled. Also, they always execute faster than real systems since waiting times can be avoided in simulators by incrementing simulation clock.

The main differences between the tools are first about the abstraction level (moderate for network simulators, high for grid ones and very high for P2P ones). Then, the achieved scale also greatly varies from tool to tool. It is worth noting that even if somehow related, scalability is not strictly correlated to abstraction, with for example GTNetS providing a better scalability than GridSim (the former was used to simulate up to 177,000 nodes by its authors – [26] – while the latter is limited to 11,000 nodes by design – [42]) with a higher level of realism.

4.5. *Comparison of Experimental Methodologies*

Tables 1 to 5 allow to compare the presented experimental methodologies through some prominent projects. If the experiment is to study existing code, the benchmark and simulation approaches are less adapted since they rely on models of the application. In-situ experiments and emulations are then more adapted (Table 1). If the studied systems is based on virtual machines (such as clouds), then in-situ methodology is well suited because it does not use models but allows direct execution of such virtual frameworks.

But if experimental control or reproducibility is the most wanted feature, the experimenter should consider emulation or simulation approaches (Tables 2 to 5). If the experiment targets extreme scales, then simulation seems more adapted since it allows to study bigger systems with process folding. For rapid prototyping, simulation seems adapted since it features faster execution time. If the experimenter wants to avoid the experimental bias due to the abstraction, s/he should avoid simulation and rely instead on emulation, or even better, on in situ experiments.

[n] http://www.buyya.com/gridsim
[o] http://simgrid.gforge.inria.fr/
[p] Note that the GES simulator also presented in [42] is a discrete time simulator and thus hardly comparable as is to SimGrid and GridSim.

Therefore, we see that each methodology has pros and cons and hence a given tool (implementing a given methodology) should be chosen with cautious depending of the experimental priority.

5. Conclusion

Computer Science is an experimental science. In many cases, experimentally test, validate or compare a proposed solution is a viable or necessary approach as the analytical approach is not always possible or sufficient. However, defining and conducting a *good experiment* (in the sense of *good practice* for an experiment) is difficult. In this paper, we have presented the role and the properties of experiments for Computer Science. We have then described four complementary methodologies (namely in-situ, emulation, benchmarking and simulation) that allow to perform *good experiments*. We have proposed a classification of these methodology depending on the level of abstraction involved (either at the application level or at the environment one). We think that such classification is very general but we have focused on large-scale systems where the experimental problematic is central. Indeed, the objects studied in this context are very hard to model and to understand analytically. 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 solutions. This survey clearly shows 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.

References

[1] 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.
[2] Peter J. Denning. ACM President's Letter: What is experimental computer science? *Commun. ACM*, 23(10):543–544, 1980.
[3] J. S. Marron, F. Hernández-Campos, and F. D. Smith. Mice and elephants visualization of internet traffic. In W. Härdle and B. Rönz, editors, *Proceedings of 15th Conference on Computational Statistics*. Physika Verlag, Heidelberg, August 2002.
[4] 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.
[5] Walter F. Tichy. Should Computer Scientists Experiment More? *Computer*, 31(5):32–40, 1998.
[6] M.V. Zelkowitz and D.R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, May 1998.
[7] Peter J. Denning. Is computer science science? *Commun. ACM*, 48(4):27–31, 2005.
[8] Peter J. Denning. ACM president's letter: performance analysis: experimental computer science as its best. *Commun. ACM*, 24(11):725–727, 1981.
[9] 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.
[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] Algorille Team, Algorithms for the Grid. INRIA Research proposal, July 2006. Available at http://www.loria.fr/equipes/algorille/algorille2.pdf.
[12] Franck Cappello and Daniel Etiemble. MPI versus MPI+OpenMP on IBM SP for the NAS benchmarks. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000.
[13] Dan Tsafrir and Dror G. Feitelson. Instability in parallel job scheduling simulation: the role of workload flurries. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island,Greece, April 2006.
[14] Vern Paxson and Sally Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–224, June 1995.
[15] The DAS-3 project: http://www.starplane.org/das3/.
[16] The Grid 5000 project: http://www.grid5000.org/.
[17] 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.
[18] Planet lab: http://www.planet-lab.org/.
[19] The xen virtual machine monitor: http://www.cl.cam.ac.uk/research/srg/netos/xen/.
[20] Qemu: open source processor emulator http://www.qemu.org/.

[21] 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.
[22] 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.
[23] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. The Nas Parallel Benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.
[24] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20, 2004.
[25] Mendel Rosenblum, Edouard Bugnion, Scott Devine, and Stephen A. Herrod. Using the simos machine simulator to study complex computer systems. In *ACM Transactions on Modeling and Computer Simulation*, volume 7, pages 78 – 103, New York, NY, USA, 1997. ACM.
[26] George F. Riley. The Georgia Tech Network Simulator. In *ACM SIGCOMM workshop on Models, Methods and Tools for Reproducible Network Research*, pages 5–12, 2003.
[27] 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.
[28] 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*, March 2008.
[29] Elliot Jaffe, Danny Bickson, and Scott Kirkpatrick. Everlab: a production platform for research in network experimentation and computation. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–11, Berkeley, CA, USA, 2007. USENIX Association.
[30] The RAMP project: Research Accelerator for Multiple Processors.
[31] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, 2002.
[32] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.
[33] iproute2+tc notes: http://snafu.freedom.org/linux2.2/iproute-notes.html.
[34] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.
[35] Jürgen Hofer and Thomas Fahringer. The Otho Toolkit – Synthesizing tailor-made scientific grid application wrapper services. *Multiagent and Grid Systems*, 3(3):281–298, 2007.
[36] 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.
[37] James H. Cowie, David M. Nicol, and Andy T. Ogielski. Modeling the global internet. *Computing in Science and Engineering*, 1(1):42–50, 1999.
[38] Stephen Naicken, Anirban Basu, Barnaby Livingston, and Sethalat Rodhetbhai. Towards yet another peer-to-peer simulator. In *Proc. Fourth International Working Conference Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs '06)*, September 2006.
[39] Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor, and Robert Rallo. Planetsim: A new overlay network simulation framework. In *Software Engineering and Middleware, SEM 2004*, volume 3437 of *LNCS*, pages 123–137, Linz, Austria, March 2005.
[40] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. PeerSim. http://peersim.sourceforge.net/.
[41] 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.
[42] Wim Depoorter, Nils Moor, Kurt Vanmechelen, and Jan Broeckhove. Scalability of grid simulators: An evaluation. In *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 544–553, Berlin, Heidelberg, 2008. Springer-Verlag.
[43] Kayo Fujiwara and Henri Casanova. Speed and Accuracy of Network Simulation in the SimGrid Framework. In *Workshop on Network Simulation Tools (NSTools)*, 2007.