A survey of general-purpose experiment management tools for distributed systems

Tomasz Buchert, Cristian Ruiz, Lucas Nussbaum, Olivier Richard

A survey of general-purpose experiment management tools for distributed systems

Tomasz Buchert[b,c,d], Cristian Ruiz[a,c,e], Lucas Nussbaum[b,c,d], Olivier Richard[a,c,e]

[a] *CNRS, LIG - UMR 5217, France*
[b] *CNRS, LORIA - UMR 7503, France*
[c] *Inria, France*
[d] *Université de Lorraine, LORIA, France*
[e] *Université Joseph Fourier - Grenoble I, LIG, France*

Abstract

In the field of large-scale distributed systems, experimentation is particularly difficult. The studied systems are complex, often nondeterministic and unreliable, software is plagued with bugs, whereas the experiment workflows are unclear and hard to reproduce. These obstacles led many independent researchers to design tools to control their experiments, boost productivity and improve quality of scientific results.

Despite much research in the domain of distributed systems experiment management, the current fragmentation of efforts asks for a general analysis. We therefore propose to build a framework to uncover missing functionality of these tools, enable meaningful comparisons between them and find recommendations for future improvements and research.

The contribution in this paper is twofold. First, we provide an extensive list of features offered by general-purpose experiment management tools dedicated to distributed systems research on real platforms. We then use it to assess existing solutions and compare them, outlining possible future paths for improvements.

1. Introduction

Distributed systems are among the most complex objects ever built by humans, as they are composed of thousands of systems that collaborate together. They also have a central role in today's society, supporting many scientific advances (scientific & high-performance computing, simulation, Big Data, etc.), and serving as the basis for the infrastructure of popular services such as Google or Facebook. Their role and popularity makes them the target of numerous research studies in areas such as scheduling, cost evaluation, fault tolerance, trust, scalability and energy consumption.

Given the size and complexity of distributed systems, it is often unfeasible to carry out analytic studies, and researchers generally use an empirical approach relying on experimentation: despite being built by humans, distributed systems are studied as if they were natural objects, with methods similar to those used in biology or physics.

One can distinguish four main methodologies for experimentation on distributed systems [1]:

- *in-situ*: a real application is tested on a real platform.
- *simulation*: a model of an application is tested on a model of the platform.

- *emulation*: a real application is tested using a model of the platform.
- *benchmarking*: a model of an application is used to evaluate a real platform.

Each methodology has its advantages and disadvantages. For example, results obtained during simulation are (usually) completely reproducible. On the other hand, as the platform is a model of the reality, the results may not apply in a general sense, as the model could lack some unnoticed but important features. In this paper we focus on experiments based on the *in-situ* and *emulation* methodologies.

Because of the actual size of the available testbeds and of the complexity of the different software layers, a lot of time is required to set up and perform experiments. Scientists are confronted with low-level tasks that they are not familiar with, making the validation of current and next generation of distributed systems a complex task. In order to lower the burden in setting up an experiment, different testbeds and experiment management tools have appeared. The last decade has seen more interest in the latter, mainly influenced by the needs of particular testbeds and other problems found in the process of experimentation such as reproducibility, replicability, automation, ease of execution, and scalability. Additionally, the existing number of papers oriented toward such tools asks for a classification in order to uncover their capabilities and limitations. Such *experiment management tools* are the main object of study in this paper. We propose a set of features that improve the experimentation process in various ways at each step (design, deployment, running the main experiment and related activities, and data and result management). Then, this list of features is used to carry out a fair comparison of tools for conducting experiments.

The rest of this paper is structured as follows. In Section 2 existing methods and approaches to experimentation with distributed systems are presented, as well as a definition of *experiment management tools*. Then, in Section 3, a set of features offered by existing experimentation tools is constructed and each element is carefully and precisely explained. In Section 4, we present a list of tools helping with research in distributed systems. Each tool is shortly presented and its features explained. Our additional observations and ideas are presented in Section 5. Finally, in Section 6 we conclude our work and discuss future work.

2. Context and terminology

This section introduces some definitions that will be used throughout this paper, as well as the context where our object of study plays its role.

2.1. Definitions

For our purposes, an *experiment* is a set of actions carried out to test (confirm, falsify) a particular hypothesis. There are three elements involved in the process: a *laboratory* (the place where one experiments), an *investigator* (the one who experiments) and an *apparatus* (the object used to measure). If an experiment can be run with a different laboratory, investigator and apparatus, and still produce the same conclusions, one says that it is *reproducible*. This is in contrast with *replicability* which requires the same results while keeping these three elements unchanged. The terms *reproducibility* and *replicability* (*replayability*) produce a lot of confusion and discrepancies as they are often used to describe different ideas and goals. The above definitions are compatible with the definitions given in [2], although we do not share such a

negative view about replicability as the authors. Being a "poor cousin" of reproducibility, replicability is nevertheless essential for the verification of results and code reusability as expressed in [3].

Finally, let us introduce a last piece of terminology and define the object of study in this paper. An *experimentation tool* or an *experiment management tool* (for research in distributed systems) is a piece of software that helps with the following main steps during the process of experimenting:

- design–by ensuring reproducibility or replicability, providing unambiguous description of an experiment, and making the experiment more comprehensible,
- deployment–by giving efficient ways to distribute files (e.g., scripts, binaries, source code, input data, operating system images, etc.), automating the process of installation and configuration, ensuring that everything needed to run the experiment is where it has to be,
- running the experiment itself–by giving an efficient way to control and interact with the nodes, monitoring the infrastructure and the experiment and signaling problems (e.g., failure of nodes),
- collection of results–by providing means to get and store results of the experiment.

Furthermore, it addresses experimentation in its full sense and it is normally conceived with one of the following purposes described fully in the next section:

- ease of experimenting,
- replicability,
- reproducibility,
- controlling and exploring parameter space.

In this study we narrow the object of study even more by considering only *general-purpose* experiment management tools (i.e., tools that can express arbitrary experimental processes) and only ones that experiment with real applications (i.e., *in-situ* and *emulation* methodologies). The former restriction excludes many tools with predefined experimental workflows whereas the latter excludes, among others, simulators (see Section 4.10).

2.2. Motivations for experimentation tools

As described before, there exist many tools that strive to ease experimentation with distributed systems. These tools are the main object of study in this article and as such they are described thoroughly in Section 4. Here, however, we discuss the main driving forces that are behind the emergence of experimentation tools.

2.2.1. Ease of experimenting

The first motivation, and the main one, for creating experimentation tools is helping with the scientific process of experimenting and making the experimenter more productive.

By providing well designed tools that abstract and outsource tedious yet already solved tasks, the development cycle can be shortened, while becoming more rigorous and targeted. Moreover, it may become more productive as the scientist may obtain additional insights and feedback that would not be available otherwise.

The ease of experimenting can indirectly help to solve the problem of research of questionable quality in the following sense. As the scientific community exerts pressure on scientists to publish more and more, they are often forced to publish results of dubious quality. If they can forget about time-consuming, low-level details of an experiment and focus on the scientific question to answer, hopefully they could spend more time testing and strengthening their results.

2.2.2. Replicability (automation)

Replicability which is also known as replayability deals with the act of repeating a given experiment under the very same conditions. In our context it means: same software, same external factors (e.g., workload, faults, etc.), same configuration, etc. If done correctly, it will lead to the same results as obtained before, allowing others to build on previous results and to carry out fair comparisons.

There are several factors that hamper this goal: size of the experiment, heterogeneity and faulty behavior of testbeds, complexity of the software stack, numerous details of the configuration, generation of repeatable conditions, etc. Among other goals, experimentation tools try to control the experiment and produce the same results under the same conditions, despite the aforementioned factors.

2.2.3. Reproducibility

It refers to the process of independent replication of a given experiment by another experimenter. Achieving reproducibility is much harder than replicability because we have to deal with the measurement bias that can appear even with the slightest change in the environment.

Therefore, in order to enhance the reproducibility of an experiment, the following features are required:

- automatic capture of the context (i.e., environment variables, command line parameters, versions of software used, software dependencies, etc.) in which the experiment is executed;
- detailed description of all the steps that led to a particular result.

The description of an experiment has to be independent of the infrastructure used. To do so abstractions for the platform have to be offered.

2.2.4. Controlling and exploring the parameter space

Each experiment is run under a particular set of conditions (parameters) that precisely define its environment. The better these conditions are described, the fuller is understanding of the experiment and obtained results. Moreover, a scientist may want to explore the *parameter space* in an efficient and adaptive manner instead of doing it exhaustively.

Typical parameters contained in a parameter space for a distributed system experiment are:

- number of nodes,
- network topology,

- hardware configuration (CPU frequency, network bandwidth, disk, etc.),
- workload during the experiment.

One can enlarge the set of parameters tested (e.g., considering CPU speed in a CPU-unaware experiment) as well as vary parameters in their allowed range (e.g., testing a network protocol under different topologies).

Whereas the capability to control the various experimental parameters can be, and quite often is, provided by an external tool or a testbed (e.g., Emulab), the high-level features helping with the design of experiments (DoE), as the efficient parameter space exploration, belong to experimentation tools.

2.2.5. Scalability

Another motivation for an experiment control is scalability of experiments, that is, being able to increase their size without harming some practical properties and scalability metrics. For example, one can consider if an experimentation tool is able to control many nodes (say, thousands) without significantly increasing the time to run the experiment, or without hampering the statistical significance of results.

The most important properties concerning scalability are:

- time–additional time needed to control the experiment (over the time to run it itself),
- resources–amount of resources required to control the experiment,
- cost of the experiment–funds required to run the experiment and control it (cf. commercial cloud computing),
- quality of results–the scientific accuracy of the results, their reproducibility in particular (contrary to the above properties, this one is hard to define and measure).

These metrics are functions of experiment parameters (see Section 2.2.4) and implementation details. Among important factors that limit scalability understood as the metrics above are:

- number of nodes used in the experiment,
- size of monitoring infrastructure,
- efficiency of data management.

2.3. Testbeds

Testbeds play an important role in the design and validation of distributed systems. They offer controlled environments that are normally shielded from the randomness of production environments. Here, we present a non-exhaustive list of testbeds that motivated the development of experiment management tools. There exists a work on defining useful features of network testbeds, similar to the goals of our paper [4]. Unsurprisingly, some features overlap in both analyses.

Grid'5000 [5] is an experimental testbed dedicated to the study of large-scale parallel and distributed systems. It is a highly configurable experimental platform with some unique features. For example, a customized operating system (e.g., with a modified kernel) can be installed and full "root" rights are available. The platform offers a REST API to control reservations, but does not provide dedicated tools to control experiments. However, the nodes can be monitored during the experiment using a simple API.

Emulab [6] is a network testbed that allows one to specify an arbitrary network topology (thanks to the emulation of the network). This feature ensures a predictable and repeatable environment for experiments. The user has access to a "root" account on the nodes and hence

can easily adjust their configuration. Emulab comes with a dedicated tool to control experiments (see 4.3).

PlanetLab [7] is a globally distributed platform for developing, deploying and accessing planetary-scale network services. It consists of geographically distributed nodes running a light, virtualized environment. The nodes are connected over the Internet. PlanetLab offers Plush (see 4.4) for the experiment control.

ORBIT [8, 9] is a radio grid testbed for scalable and reproducible evaluation of next-generation wireless network protocols. It offers a novel approach involving a large grid of radio nodes which can be dynamically interconnected into arbitrary topologies with reproducible wireless channel models. A dedicated tool to run experiments with ORBIT platform is OMF (see 4.6).

DAS[1] (Distributed ASCI Supercomputer) is a Dutch wide-area distributed system designed by the Advanced School for Computing and Imaging (ASCI). Distinguishably, it employs various HPC accelerators (e.g., GPUs) and novel network interconnect. Its most recent iteration is DAS-4. DAS does not offer a dedicated tool to control experiments, however it provides a number of tools to help with deployment, discovering problems and scheduling.

With the emergence of efficient and cheap virtualization, the scientists turn to *cloud computing infrastructures* as a viable experimentation platform. A popular commercial service is Amazon EC2[2], but many alternatives and variations exist (e.g., Windows Azure[3]). There are non-commercial, open-source solutions available as well (e.g., OpenStack[4]). Even though the development of cloud computing solutions was not inspired by a need of a research platform, the scalability and elasticity offered by those make it an attractive solution for science. In [10] a framework oriented toward reproducible research on such infrastructures is proposed.

3. List of features offered by experiment management tools

In this section, we present properties available in experiment management tools for distributed systems after doing a literature review using the following sources:

- tools used and published by the most important and large-scale testbeds (see Section 2.3),
- papers referenced by these tools and papers that cite them,
- IEEE and ACM digital libraries search with the following keywords in the abstract or title: *experiments*, *experiment*, *distributed systems*, *experimentation*, *reproducible*.

We ended up with 8 relevant tools for managing experiments that met our criteria of an *experimentation tool*, however we also include *Naive approach* (see Section 4.1) in our analysis. An extensive analysis of the papers dedicated to those tools was performed; subsequently, a set of properties and features - highlighted by each of the tools as to be important for the experimentation process - was selected and classified. The list consists of nine groups of properties and features that have an important role in the experimentation process. The complete hierarchy is presented in Figure 1.

[1] `http://www.cs.vu.nl/das4/`
[2] `http://aws.amazon.com/ec2/`
[3] `http://www.windowsazure.com/`
[4] `http://www.openstack.org/`

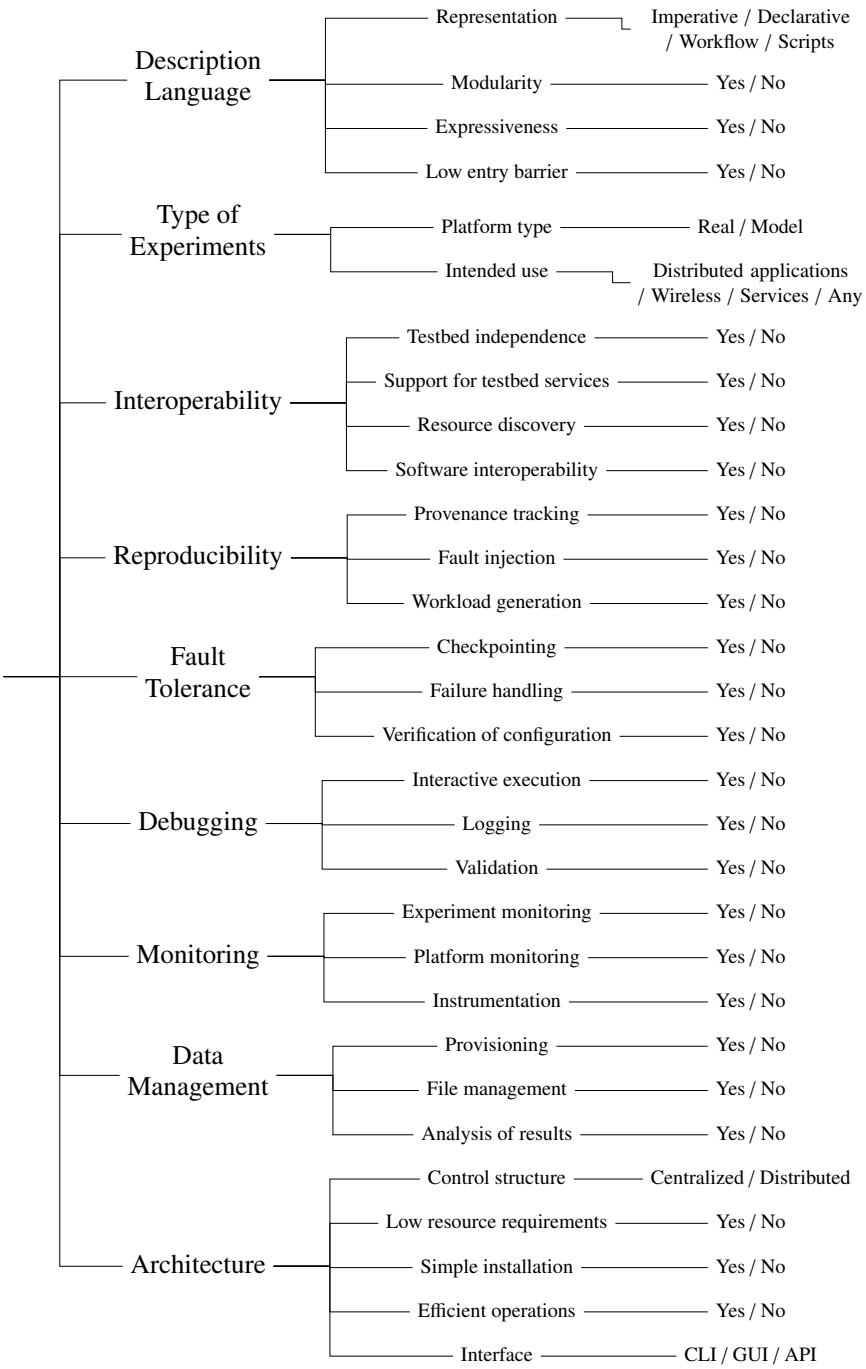

Figure 1: The tree of features. All evaluated properties and features are presented with their respective domains of values. The properties are grouped into 9 groups that cover different aspects of experiment management.

3.1. Description Language

The design of the experiment is the very first step in the experimentation process. The description language helps users with this step, allowing them to describe how the experiment has to be performed, as well as their needs for running the experiment. Characteristics that help with describing the experiment are presented in the following sections.

REPRESENTATION (IMPERATIVE / DECLARATIVE / WORKFLOW / SCRIPTS) of experiments featured by a given tool is the approach used to describe the experiment and relevant details. Possible representations differ in their underlying paradigm (e.g., imperative, declarative) and in a level of abstraction that the description operates on. Some tools use low-level scripts to build experiments whereas others turn to higher abstractions, some of them graphical (e.g., workflows). The choice of a certain representation has implications on other aspects of the description language.

MODULARITY (YES / NO) is a property of experiment description language that enables easy adding, removing, replacing and reusing parts of experiments. An experiment expressed in a modular way can be logically split into modules with well-defined interfaces that can be worked on independently, possibly by different researchers specializing in a particular aspect of the experiment.

EXPRESSIVENESS (YES / NO) that makes it effective in conveying thoughts and ideas, in short and succinct form. Expressiveness provides a more maintainable, clearer description. Various elements can improve expressiveness: well-chosen abstractions and constructions, high-level structure, among others.

LOW ENTRY BARRIER (YES / NO) is the volume of work needed to switch from naive approach to the given approach while assuming prior knowledge about the infrastructure and the experiment itself. In other words, it is the time required to learn how to efficiently design experiments in the language of the given experimentation tool.

3.2. Type of Experiments

This encompasses two important aspects of an experiment: the platform where the experiments are going to be run on and the research fields where those experiments are performed.

PLATFORM TYPE (REAL / MODEL) is the range of platforms supported by the experimentation tool. The platform type can be *real* (i.e., consists of physical nodes) or be a *model* (i.e., built from simplified components that model details of the platform like network topology, links bandwidth, CPU speed, etc.). For example, platforms using advanced virtualization or emulation techniques (like Emulab testbed) are considered to be modeled. Some testbeds (e.g., PlanetLab) are considered real because they do not hide the complexity of the platform, despite the fact that they use virtualization.

INTENDED USE (DISTRIBUTED APPLICATIONS / WIRELESS / SERVICES / ANY) refers to the research context the experimentation tool targets. Examples of research domains that some tools specialize in include: wireless networks, network services, high performance computing, peer-to-peer networks, among many others.

3.3. Interoperability

It is important for an experimentation tool to interact with different platforms, as well as to exploit their full potential. The interaction with external software is an indisputable help during the process of experimenting.

TESTBED INDEPENDENCE (YES / NO) of the experimentation tool is its ability to be used with different platforms. The existing tools are often developed along with a single testbed and tend to focus on its functionality and, therefore, cannot be easily used somewhere else. Other tools explicitly target a general use and can be used with a wide range of experimental infrastructures.

SUPPORT FOR TESTBED SERVICES (YES / NO) is a capability of the tool to interface different services provided by the testbed where it is used (e.g., resource requesting, monitoring, deployment, emulation, virtualization, etc.). Such a support may be vital to perform scalable operations efficiently, exploit advanced features of the platform or to collect data unavailable otherwise.

RESOURCE DISCOVERY (YES / NO) is a feature that allows to reserve a set of testbed resources meeting defined criteria (e.g., nodes with 8 cores interconnected with 1 Gbit network). Among methods to achieve this feature are: interoperating with testbed resource discovery services or emulation of resources by the tool.

SOFTWARE INTEROPERABILITY (YES / NO) is the ability of using various types of external software in the process of experimenting. The experimentation tool that interoperates with software should offer interfaces or means to access or integrate monitoring tools, commands executers, software installers, package managers, etc.

3.4. Reproducibility

This group concerns all methods used to help with reproducibility and repeatability as was described in Section 2.2.3.

PROVENANCE TRACKING (YES / NO) is defined as a way of tracing and storing information of how scientific results have been obtained. An experimentation tool supports data provenance if it can describe the history of a given result for a particular experiment. An experimentation tool can provide data provenance through the tracking of details at different layers of the experiment. At a low-level layer, the tool must be able to track details such as: command-line parameters, process arguments, environment variables, version of binaries, libraries and kernel modules in use, hardware devices used, and file system operations executed. At a high-level layer, it must track details such as: number of nodes used, details of used machines, timestamps of operations, and state of the platform.

FAULT INJECTION (YES / NO) is a feature that enables the experimenter to introduce factors that can modify and disrupt the functioning of the systems being studied. These factors include: node failures, link failures, memory corruption, background CPU load, etc. This feature allows to run experiments under more realistic and challenging conditions and test behavior of the studied system under exceptional situations.

WORKLOAD GENERATION (YES / NO) is a range of features that allow to inject a predefined workload into the experimental environment (e.g., number of requests to a service). The generated workload is provided by real traces or by synthetic specification. Similarly to fault injection, this feature allows to run experiments in more realistic scenarios.

3.5. Fault Tolerance

This group of features encompasses all of them that help with common problems that can happen during experiments and may lead to either invalid results (especially dangerous if gone unnoticed) or to increased time required to manually cope with them.

CHECKPOINTING (YES / NO) allows to save a state of the experiment and to restore it later as if nothing happened. It is a feature that can, above all, save the time of the user. There are at least two meanings of checkpointing in our context:

- only some parts of the experiment are saved or cached,
- the full state of the experiment is saved (including the platform).

Of course, the second type of checkpointing is much more difficult to provide. Checkpointing helps with fault tolerance as well, since a failed experiment run will not necessarily invalidate the whole experiment.

FAILURE HANDLING (YES / NO) of the experimentation tool can mitigate runtime problems with the infrastructure an experiment is running on. This means in particular that failures are detected and appropriate steps are taken (e.g., the experiment is restarted). Typical failures are crashing nodes, network problems, etc.

VERIFICATION OF CONFIGURATION (YES / NO) consists in having an automatic way to verify the state of an experimentation platform. Usually such a step is performed before the main experiment to ensure that properties of the platform agree with a specification. We distinguish verification of:

- software – ensuring that the software is coherent on all computing nodes,
- hardware – ensuring that the hardware configuration is as it is supposed to be.

3.6. Debugging

The features grouped in this section help to find problems and their causes during the experimentation process.

INTERACTIVE EXECUTION (YES / NO) refers to an ability to run the experiment "on-the-fly" including: manually scheduling parts of the experiment, introspecting its state and observing intermediate results. This feature is inspired by debuggers offered by integrated development environments (IDEs) for programming languages.

LOGGING (YES / NO) consists of features that allow bookkeeping of low-level messages emitted during experiments including those that were placed at arbitrary places by the experimenter. The messages are normally stored sequentially along with their timestamps making the log is essentially a one-dimensional dataset. The log can be used to debug an experiment and document its execution.

VALIDATION (YES / NO) is a feature that offers the user a way to perform a fast (that is, faster than full execution of the experiment) and automatic way to verify the description of an experiment. Depending on the modeling language used and other details, the validation may be accordingly thorough and complete. For our purposes, we require that at least some semantic analysis must be performed, in contrast to simple syntactic analysis.

3.7. Monitoring

Monitoring is necessary to understand the behavior of the platform and the experiment itself. It consists in gathering data from various sources: the experiment execution information, the platform parameters and metrics, and other strategic places like instrumented software.

EXPERIMENT MONITORING (YES / NO) consists in observing the progress of the experiment understood as set of timing and causal information between actions in the experiment. The monitoring includes keeping track of currently running parts of the experiment as well as their interrelations. Depending on the model used, this feature may take different forms.

PLATFORM MONITORING (YES / NO) is the capability of an experimentation tool to know the state of resources that comprise the experiment (nodes, network links, etc.). Data collected that way may be used as a result of the experiment, to detect problems with the execution or as a way to get additional insights about the experiment.

INSTRUMENTATION (YES / NO) enables the user to take measurements at different moments and places while executing the experiment. This includes instrumentation of software in order to collect measures about its behavior (CPU usage, performance, resource consumption, etc.).

3.8. Data Management

The management of data is an important part of the experiment. This section contains features that help with distribution and collection of data.

PROVISIONING (YES / NO) is the set of actions to prepare a specific physical resource with the correct software and data, and make it ready for the experimentation. Provisioning involves tasks such as: loading of appropriate software (e.g., operating system, middleware, applications), configuration of the system and starting necessary services. It is necessary for any experimentation tool to provide at least a rudimentary form of this functionality.

FILE MANAGEMENT (YES / NO) is a feature that abstracts a tedious job of working with files. Therefore the user does not have to manage them manually at a low level which often is error-prone. This includes actions like automatic collection of results stored at participating nodes.

ANALYSIS OF RESULTS (YES / NO) is a service of an experimentation tool that is used to collect, store and visualize experimental results, as well as making dynamic decisions based on their runtime values. The latter ability paves a way for intelligent design of experiments by exploring only relevant regions of parameter space and therefore saving resources like energy or time.

3.9. Architecture

This section contains features and properties related to how the tool is designed and what architecture decisions the authors made. This includes ways to interact with the tool, as well as technical details such as software dependencies, methods to achieve scalability and efficient execution of experiments.

CONTROL STRUCTURE (CENTRALIZED / DISTRIBUTED) refers to the structure of nodes used to control the experiment. The architecture of a tool is *centralized* if the control of an experiment is centralized and there exists one node that performs all principal work. Otherwise, if there are multiple nodes involved in the experiment control, then the architecture is *distributed*.

LOW RESOURCE REQUIREMENTS (YES / NO) of an experimentation tool refer to its resource consumption (memory, CPU, network bandwidth, etc.) associated with the activity of controlling the experiment. As the number of elements the experiment consists of increases (e.g., nodes), so does the amount of the resources necessary to control them.

SIMPLE INSTALLATION (YES / NO) is understood as a low difficulty of setting up a completely functional infrastructure that the tool needs in order to be used. This usually implies software dependencies (interpreters, libraries, special services, etc.) or a required hardware infrastructure (number of network interfaces, minimum memory size, number of dedicated nodes to control the experiment, etc.)

EFFICIENT OPERATIONS (YES / NO) is the range of features that provide methods, tools and algorithms to perform large-scale operations with the experimental infrastructure. This in particular includes: efficient and scalable methods for command execution, file distribution, monitoring of nodes, gathering of results, among others. Providing efficient versions of these actions is notably difficult as operations involving nodes in a distributed systems are non-trivially scalable as a number of nodes increases.

INTERFACE (CLI / GUI / API) consists of different ways that the user can interact with the experimentation tool. Most of the tools provide command line interface, whereas some tools provide graphical interfaces, usually via webpage used to interact with the experiment.

4. Existing experimentation tools

The aim of this section is to present the state of the art of the existing tools for experimentation with distributed systems. We focus our attention on the tools that fulfill the criteria for being considered as an experimentation tool (for a list of tools that are not included in the analysis, see Section 4.10). The evaluation of all tools and the main result of our study are presented in Table 1 that shows a comparison of the tools based on the proposed list of features. Figure 2 shows a timeline of publications about these experiment management tools whereas Table 2 shows their impact measured as the number of citations about them.

4.1. Naive method

Frequently, experiments are done using this method which includes manual procedures and use of hand-written and low-level scripts. Lack of modularity and expressiveness is commonly seen because of the *ad hoc* nature of these scripts, and it is even worse when the experiment involves many machines. The experiment is controlled at a very low level, including some human

		Naive approach	Weevil	Workbench	Plush/Gush	Expo	OMF	NEPI	XPFlow	Execo
Description Language (18/27 ≈ 67%)	Representation	Scripts	Declarative[12]	Imperative[13]	Declarative[14]	Imperative[15]	Imperative[16]	Imperative[17]	Declarative[18]	Imperative[19]
	Modularity (4/9)	No	Yes	No	No	No	No	Yes	Yes	Yes
	Expressiveness (7/9)	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
	Low entry barrier (7/9)	Yes	No	Yes	Yes[20]	Yes	Yes	Yes	No	Yes
Type of Experiments	Platform type	Real	Real	Model	Real	Real	Real	Real, Model	Real	Real
	Intended use	Any	Services	Any	Any	Any	Wireless[21]	Any	Any	Any
Interoperability (22/36 ≈ 61%)	Testbed independence (8/9)	Yes	Yes	No	Yes[22]	Yes	Yes	Yes	Yes	Yes
	Support for testbed services (7/9)	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Resource discovery (5/9)	No	No	Yes⋆	Yes	Yes⋆	Yes	Yes	No	No
	Software interoperability (2/9)	No	No	No	Yes	No	Yes	No	No	No
Reproducibility (4/27 ≈ 15%)	Provenance tracking (1/9)	No	No	Yes	No	No	No	No	No	No
	Fault injection (2/9)	No	Yes	No	No	No	Yes⋆	No	No	No
	Workload generation (1/9)	No	Yes	No	No	No	No	No	No	No
Fault Tolerance (12/27 ≈ 44%)	Checkpointing (4/9)	No	Yes	No	No	No	No	Yes	Yes	Yes
	Failure handling (6/9)	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes
	Verification of configuration (2/9)	No	No	Yes⋆	No	No	Yes	No	No	No
Debugging (17/27 ≈ 63%)	Interactive execution (7/9)	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes
	Logging (6/9)	No	No	Yes⋆	No	Yes	Yes	Yes	Yes	Yes
	Validation (4/9)	No	Yes	Yes	No	No	No	Yes	Yes	No
Monitoring (10/27 ≈ 37%)	Experiment monitoring (4/9)	No	No	Yes	No	No	Yes	Yes	Yes	No
	Platform monitoring (4/9)	No	No	Yes⋆	Yes	No	Yes	Yes	No	No
	Instrumentation (2/9)	No	No	No	Yes	No	Yes	No	No	No
Data Management (13/27 ≈ 48%)	Provisioning (5/9)	No	Yes	Yes⋆	Yes	No	Yes	Yes	No	No
	File management (5/9)	No	Yes	Yes	Yes	No	Yes	No	No	Yes
	Analysis of results (3/9)	No	No	Yes	No	No	Yes	No	Yes	No
Architecture (19/27 ≈ 70%)	Control structure	Centralized	Centralized	Centralized	Centralized	Centralized	Distributed	Distributed	Centralized	Centralized
	Low resource requirements (6/9)	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes
	Simple installation (7/9)	Yes	Yes	No	Yes	Yes	No	Yes	Yes	Yes
	Efficient operations (6/9)	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes
	Interface	CLI	CLI	GUI, CLI, API	CLI, GUI, API	CLI	CLI, GUI	CLI, GUI	CLI	CLI

[1] GNU m4
[2] Event-based (Tcl & ns)
[3] XML
[4] Ruby
[5] Event-based (Ruby)
[6] Modular API based on Python
[7] Workflows (Ruby)
[8] Modular API based on Python
[9] Using GUI
[10] Supports wired resources as well
[11] PlanetLab oriented
[12] GNU m4
[13] Event-based (Tcl & ns)
[14] XML
[15] Ruby
[16] Event-based (Ruby)
[17] Modular API based on Python
[18] Workflows (Ruby)
[19] Modular API based on Python
[20] Using GUI
[21] Supports wired resources as well
[22] PlanetLab oriented
⋆ Provided by testbed

Table 1: Summary of analyzed experiment management tools for distributed systems research. Each feature is presented along with a number of tools that provide it. Similarly, for each group a percentage of implemented features from this group is shown. Features that are due to the integration with a testbed are marked with ⋆.

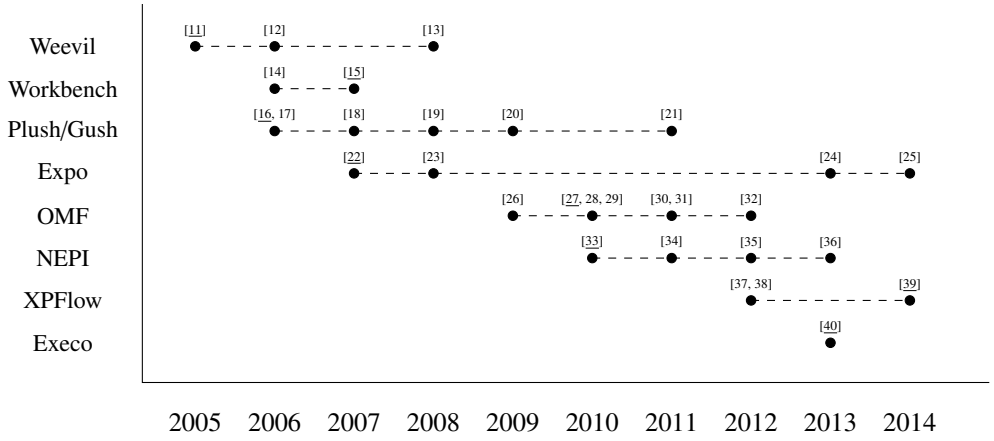

Figure 2: Timeline of publications dedicated to experiment management tools. The publication that attracted most of the citations (main publication) is underlined.

intervention. Therefore, interaction with many types of applications and platforms is possible at the cost of time required to do so. Parameters for running the experiment can be forgotten as well as the reason for which they were used. This leads to an experiment that is difficult to understand and repeat. Since the experiment is run in partially manual fashion, the user can react against some unexpected behaviors seen during the experiment.

4.2. Weevil

It is a tool to evaluate distributed systems under real conditions, providing techniques to automate the experimentation activity. This experimentation activity is considered as the last stage of development. Experiments are described declaratively with a language that is used to instantiate various models and provides clarity and expressiveness. Workload generation is one of its main features, which helps with the replicability of results.

4.3. Workbench for Emulab

Workbench is an integrated experiment management system, which is motivated by the lack of replayable research on the current testbed-based experiments. Experiments are described using an extended version of the *ns* language which is provided by Emulab. The description encompasses static definitions (e.g., network topology, configuration of devices, operating system and software, etc.) and dynamic definitions of activities that are based on *program agents*, entities that run programs as part of the experiment. Moreover, activities can be scheduled or can be triggered by defined events. Workbench provides a generic and parametric way of instantiating an experiment using features already provided by Emulab to manage experiments. This allows experimenters to run different instances of the same experiment with different parameters. All pieces of information necessary to run the experiment (e.g., software, experiment description, inputs, outputs, etc.) are bundled together in *templates*.

Templates are both persistent and versioned, allowing experimenters to move through the history of the experiment and make comparisons. Therefore, the mentioned features facilitate the replay of experiments, reducing the burden on the user. Data management is provided by the

underlying infrastructure of Emulab, enabling Workbench to automatically collect logs that were generated during the experiment.

4.4. Plush/Gush

Plush, and its another incarnation called Gush, cope with the deployment, maintenance and failure management of different kinds of applications or services running on PlanetLab. The description of the application or services to be controlled is done using *XML*. This description comprehends the acquisition of resources, software to be installed on the nodes and the workflow of the execution. It has a lightweight client-server architecture with a few dependencies that can be easily deployed on a mix of normal clusters and GENI control frameworks: PlanetLab, ORCA[5] and ProtoGENI[6]. One of the most important features of *Plush* is its capacity to manage failures. The server receives a constant stream of information from all the client machines involved in the experiment and performs corrective actions when a failure occurs.

4.5. Expo

Expo offers abstractions for describing experiments, enabling users to express complex scenarios. These abstractions can be mapped to the hierarchy of the platform or can interface underlying tools, providing efficient execution of experiments. Expo brings the following improvements to the experimentation activity: it makes the description of the experiment easier and more readable, automates the experimentation process, and manages experiments on a large set of nodes.

4.6. OMF

It is a framework used in different wireless testbeds around the world and also in PlanetLab. Its architecture versatility aims at federation of testbeds. It was mainly conceived for testing network protocols and algorithms in wireless infrastructures. The OMF architecture consists of 3 logical planes: Control, Measurement, and Management. Those planes provide users with tools to develop, orchestrate, instrument and collect results as well as tools to interact with the testbed services. For describing the experiment, it uses a comprehensive domain specific language based on Ruby to provide experiment-specific commands and statements.

4.7. NEPI

NEPI is a Python library that enables one to run experiments for testing distributed applications on different testbeds (e.g., PlanetLab, OMF wireless testbeds, network simulator, etc). It provides a simple way for managing the whole experiment life cycle (i.e., deployment, control and results collection). One important feature of NEPI is that it enables to use resources from different platforms at the same time in a single experiment. NEPI abstracts applications and computational equipment as resources that can be connected, interrogated and conditions can be registered in order to specify workflow dependencies between them.

[5] `http://groups.geni.net/geni/wiki/ORCABEN`
[6] `http://www.protogeni.net`

4.8. XPFlow

XPFlow is an experimentation tool that employs *business workflows* in order to model and run experiments as *control flows*. XPFlow serves as a workflow engine that uses a domain-specific language to build complex *processes* (experiments) from smaller, independent tasks called *activities*. This representation is claimed to bring useful features of Business Process Modeling (BPM), that is: easier understanding of the process, expressiveness, modularity, built-in monitoring of the experiment, and reliability.

Both XPFlow and scientific workflow systems rely on workflows. However, scientific workflows are data-oriented and the distributed system underneath (e.g., a computational grid) is merely a tool to efficiently process data, not an object of a study. Moreover, the formalism of XPFlow is inspired by workflow patterns identified in the domain of BPM, which are used to model *control flows*, as opposed to *data flows* (see Section 4.10.2).

4.9. Execo

Execo is a generic toolkit for scripting, conducting and controlling large-scale experiments in any computing platform. Execo provides different abstractions for managing local and remote processes as well as files. The engine provides functionality to track the experiment execution and offers features such as *parameter sweep* over a defined set of values. The partial results of the parameter sweep can be saved to persistent storage, therefore avoiding unnecessary reruns in case of a failure.

4.10. Tools not covered in the study

In the following section, we discuss other tools that could be mistaken as an experiment management tool according to our definition. They either contradict the definition (cf. Section 4.10.1) or support only subset of all activities required by it (cf. Section 4.10.4). The latter ones are sometimes used by experiment management tools to implement features presented in Section 3.

4.10.1. Non general-purpose experiment management tools

Tools like ZENTURIO [41] and Nimrod [42] help experimenters to manage the execution of parameter studies on cluster and Grid infrastructures. Both tools cover activities like the set up of the infrastructure to use, collection and analysis of results. ZENTURIO offers a more generic parametrization, making it suitable for studying parallel applications under different scenarios where different parameters can be changed (e.g., application input, number of nodes, type of network interconnection, etc.). Even though Nimrod parametrization is restricted to application input files, a relevant feature is the automation of the design of fractional factorial experiments. NXE [43] scripts the execution of several steps of the experimental workflow from the reservation of resources in a specific platform to the analysis of collected logs. The whole experiment scenario is described using XML which is composed of three parts: topology, configuration and scenario. All the interactions with resources and applications are wrapped using bash scripts. NXE is mainly dedicated to the evaluation of network protocols.

The aforementioned tools were not included in our analysis, because there are not *general-purpose* experiment management tools. They address only very specific scenarios of experimentation with a distributed system like parameter studies and network protocols evaluation.

4.10.2. Scientific workflow systems

The aim of scientific workflow systems is automation of the scientific process that a scientist may go through to get publishable results from raw data. The main objective is to communicate analytical procedures repeatedly with minimal effort, enabling the collaboration on conducting large, data-processing, scientific experiments. Scientific workflows are designed specifically to compose and execute a series of computational or data manipulation steps. Normally, those systems are provided with GUIs that enable non-expert users to easily construct their applications as a visual graph. Goals such as data provenance and experiment repeatability are both shared by scientific workflows and experimentation tools. Some examples of scientific workflows are: Kepler [44], Taverna [45] and Vistrails [46]. An interesting analysis of these systems, and a motivation for this work, is presented in [47].

There are two main reasons why scientific workflows are not covered in our study. First, scientific workflows are *data flows* in nature–they are used to run complex computations on data, while the computational platform is abstracted and the user has no direct control over it (e.g., the nodes used during computation). Hence the platform is not the object of study, but merely a tool to carry out computation. Second, the declarative representation of many scientific workflows as acyclic graphs is generally limited in its expressiveness, therefore they do not meet the criteria of *general-purpose* experimentation tools according to our definition (see [48, 49] for analyses of scientific workflows expressiveness).

4.10.3. Simulators and abstract frameworks

An approach widely used for evaluating and experimenting with distributed systems is simulation. In [50] the most used simulators for overlay networks and peer-to-peer applications are presented. Another framework called SimGrid [51] is used for the evaluation of algorithms, heuristics and even real MPI applications in distributed systems such as Grid, Cloud or P2P systems.

Even though simulators provide many features required by the definition of the experimentation tool, they are not included in our study. First, they do not help with experiments on real platforms as they provide an abstract and modeled platform instead. Second, the goals of simulators are often very specific to a particular research subdomain and hence are not general-purpose tools [51].

Other tools such as Splay [52] and ProtoPeer [53] go one step further by making easy the transition between simulation and real deployment. Both tools provide a framework to write distributed applications based on a model of the target platform. They are equipped with measurement infrastructures and event injection for reproducing the dynamics of a live system.

The tools providing abstract framework to write applications under experimentation are not considered in our study, because real applications cannot be evaluated with them. Although real machines may be used to run experiments (as it is the case with Splay), the applications must be ported to APIs provided by these tools.

4.10.4. Configuration and orchestration management software

Puppet[7] and Chef[8] are commonly used in automating administrative tasks such as software provision and configuration of operating systems. They simplify complex deployments by pro-

[7] `https://puppetlabs.com/`

[8] `http://www.opscode.com/chef/`

Tool	First publication	Citations
Weevil	2005	69
Workbench	2006	80
Plush/Gush	2006	177
Expo	2007	16
OMF	2009	152
NEPI	2010	38
XPFlow	2012	3
Execo	2013	1

Table 2: Number of publications citing papers dedicated to each experimentation tool (as verified on 4 July 2014).

viding unambiguous, declarative description of a desired system state and then carrying out necessary steps to reach it. Operating at even higher level are orchestration management tools, like Juju[9], which are designed to coordinate complex systems in flexible and reactive ways, usually in the cloud computing context.

All these tools do not fulfill the definition of the experiment management tool. First, they are not *general-purpose* since no precise control over the execution is available (which is actually the goal of these tools). Second, the collection of results is not present.

4.10.5. Tools capturing experimental context

As mentioned in Section 2.2.3 one important feature required given the complexity of software nowadays, is the capture of the experimental context, undoubtedly useful for the reproduction of an experiment. Experimenters can take advantage of version control systems (e.g., Git, Subversion) or more sophisticated frameworks like Sumatra [3] which aim at recording and tracking the scientific context in which a given experiment was performed.

These tools are not experiment management tools according to our definition, of course, yet their use is convenient to document history of any software project, including experiment description and results. Some tools use them as a building block to store experimental context (cf. Section 4.3).

5. Discussion

Existing tools for experiment control were analyzed and evaluated using our set of features defined in Section 3 and the final results are presented in Table 1. For each position in the table (i.e., each property/tool pair) we sought for an evidence to support possible values of a given property in a given tool from a perspective of a prospective user. To this end, the publications, documentation, tutorials and other on-line resources related to the given approach were consulted. If presence of the property (or lack thereof) could be clearly shown from these observations, the final value in the table reflects this fact. However, if we could not find any mention

[9] `https://juju.ubuntu.com/`

of the feature, then the final value claims that the feature does not exist in the tool, as for all practical purposes the prospective user would not be aware of this feature, even if it existed. In ambiguous cases additional comments were provided. Much more detailed analysis that led to this concise summary is available on-line[10]. Using information collected in the table, one can easily draw few conclusions.

There is no agreement whether a declarative description is more beneficial than an imperative one. Declarative descriptions seem to be associated with higher modularity and expressiveness, but at a price of a higher entry barrier. Moreover, the tools tend to be independent of a particular testbed, but those with tight integration offer a more complete set of features or features not present in other solutions (e.g., Emulab Workbench).

The majority of addressed features come from *Architecture* (70%), *Description Language* (67%), *Debugging* (63%) and *Interoperability* (61%) groups. On the other hand, support for *Fault Tolerance* and *Monitoring* is quite low (44% and 37%, respectively), whereas support for *Reproducibility* is almost nonexistent (only 15%). The features available in majority of the analyzed tools are: *Testbed independence* (8/9), *Expressiveness* (7/9), *Low entry barrier* (7/9), *Support for testbed services* (7/9), *Interactive execution* (7/9), *Failure handling* (6/9), *Logging* (6/9), *Resource discovery* (5/9), *File management* (5/9) and *Provisioning* (5/9). Moreover, the tools have nearly universally *Simple installation* (7/9), *Low resource requirements* (6/9) and offer methods to perform *Efficient operations* (6/9). The two most unimplemented features are *Provenance tracking* (1/9) and *Workload generation* (1/9), both crucial for reproducibility of experiments.

Additionally, some tools offer unique features: *Software interoperability* (Plush and OMF), *Provenance tracking* (Workbench), *Fault injection* (Weevil and OMF), *Workload generation* (Weevil), *Verification of configuration* (Workbench and OMF) and *Instrumentation* (Plush and OMF). However, it is worth pointing out that features such as *Workload generation* are often provided by standalone tools.

Finally, we did a simple "impact analysis" of described tools by summing all unique scientific citations to papers about each tool using Google Scholar (see Table 2). Clearly, without adjusting the score to the age of each tool, the most cited tool is Plush. As interesting as these data may be, we abstain from drawing any more conclusions from them. The summary of this analysis is available on-line[11].

6. Conclusions

In this paper, we presented an extensive list of properties expected from general-purpose experiment management tools for distributed systems on real platforms. The diversity of the research domain of distributed systems motivated development of different techniques and tools to control experiments, and explains the multitude of approaches to manage experiments. With the construction of the feature list, we tried to establish a common vocabulary in order to understand and compare the existing experiment management tools.

The size and complexity of distributed systems nowadays have uncovered new concerns and needs in the experimentation process. We need to control an always increasing number of variables to assure two important characteristics of an experiment, its reproducibility and replicability. With the motivation of providing a controlled environment to execute experiments in the

[10] `http://www.loria.fr/~buchert/exp-survey.yaml`
[11] `http://www.loria.fr/~buchert/exp-impact.yaml`

domain of distributed systems, several testbeds were created which stimulated the development of different experiment management tools. Among the benefits of experiment management tools are: encouraging researchers to experiment more and improve their results, educational value of being able to play with known algorithms and protocols under real settings, reduction of the time required to perform an evaluation and publish results, capacity to experiment with many nodes and complex scenarios, different software layers, topologies, workloads, etc.

Despite the emergence of experiment management tools, some of them are in an immature state of development which prevents them from fully exploiting the capacity of certain testbeds. There is indeed, a lot of challenges in the domain of experimentation and the need of further development of those tools is apparent. To achieve this, technologies developed with different purposes could arguably be used in the experimentation process. For instance, we mentioned that workflow systems and configuration management tools share some concerns and goals with the problem of experimenting with distributed systems.

Finally, a deeper understanding of the experimentation process with distributed systems is needed to identify novel ways to perfect the quality of experiments and give researchers the possibility to build on each others' results.

References

[1] J. Gustedt, E. Jeannot, M. Quinson, Experimental Methodologies for Large-Scale Systems: a Survey, Parallel Processing Letters 19 (2009) 399–418.

[2] C. Drummond, Replicability is not reproducibility: Nor is it good science, in: Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML, p. 4972–4975.

[3] A. Davison, Automated Capture of Experiment Context for Easier Reproducibility in Computational Research, Computing in Science and Engg. 14 (2012) 48–56.

[4] C. Siaterlis, M. Masera, A survey of software tools for the creation of networked testbeds, International Journal On Advances in Security 3 (2010) 1–12.

[5] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jégou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, G. Mornet, Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform, in: 6th IEEE/ACM International Workshop on Grid Computing (Grid), pp. 99–106.

[6] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar, An Integrated Experimental Environment for Distributed Systems and Networks, in: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), USENIX Association, Boston, MA, pp. 255–270.

[7] L. Peterson, T. Anderson, D. Culler, T. Roscoe, A blueprint for introducing disruptive technology into the Internet, SIGCOMM Comput. Commun. Rev. 33 (2003) 59–64.

[8] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, M. Singh, Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols, in: Wireless Communications and Networking Conference, 2005 IEEE, volume 3, pp. 1664–1669.

[9] M. Ott, I. Seskar, R. Siraccusa, M. Singh, ORBIT testbed software architecture: supporting experiments as a service, in: Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on, pp. 136–145.

[10] J. Klinginsmith, M. Mahoui, Y. M. Wu, Towards Reproducible eScience in the Cloud, in: 3rd IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM), pp. 582–586.

[11] Y. Wang, M. J. Rutherford, A. Carzaniga, A. L. Wolf, Automating Experimentation on Distributed Testbeds, in: Proceedings of the 20th IEEE/ACM International Conference On Automated Software Engineering (ASE), ASE '05, ACM, New York, NY, USA, 2005, pp. 164–173.

[12] Y. Wang, Automating experimentation with distributed systems using generative techniques, Ph.D. thesis, University of Colorado at Boulder, Boulder, CO, USA, 2006. AAI3219040.

[13] Y. Wang, A. Carzaniga, A. L. Wolf, Four enhancements to automated distributed system experimentation methods, in: Proceedings of the 30th international conference on Software engineering, ICSE '08, ACM, New York, NY, USA, 2008, pp. 491–500.

[14] E. Eide, L. Stoller, T. Stack, J. Freire, J. Lepreau, Integrated scientific workflow management for the Emulab network testbed, in: Proceedings of the annual conference on USENIX '06 Annual Technical Conference, ATEC '06, USENIX Association, Berkeley, CA, USA, 2006, pp. 33–33.

[15] E. Eide, L. Stoller, J. Lepreau, An Experimentation Workbench for Replayable Networking Research, in: Proceedings of the 4th Symposium on Networked System Design and Implementation (NSDI), pp. 215–228.
[16] J. Albrecht, C. Tuttle, A. C. Snoeren, A. Vahdat, PlanetLab Application Management Using PluSH, ACM SIGOPS Operating Systems Review 40 (2006) 33–40.
[17] J. Albrecht, C. Tuttle, A. C. Snoeren, A. Vahdat, Loose synchronization for large-scale networked systems, in: Proceedings of the annual conference on USENIX '06 Annual Technical Conference, ATEC '06, USENIX Association, Berkeley, CA, USA, 2006, pp. 28–28.
[18] J. Albrecht, R. Braud, D. Dao, N. Topilski, C. Tuttle, A. C. Snoeren, A. Vahdat, Remote control: distributed application configuration, management, and visualization with Plush, in: Proceedings of the 21st conference on Large Installation System Administration Conference, LISA'07, USENIX Association, Berkeley, CA, USA, 2007, pp. 15:1–15:19.
[19] N. Topilski, J. Albrecht, A. Vahdat, Improving scalability and fault tolerance in an application management infrastructure, in: First USENIX Workshop on Large-Scale Computing, LASCO'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 2:1–2:12.
[20] J. R. Albrecht, Bringing big systems to small schools: distributed systems for undergraduates, in: Proceedings of the 40th ACM technical symposium on Computer science education, SIGCSE '09, ACM, New York, NY, USA, 2009, pp. 101–105.
[21] J. Albrecht, C. Tuttle, R. Braud, D. Dao, N. Topilski, A. C. Snoeren, A. Vahdat, Distributed Application Configuration, Management, and Visualization with Plush, ACM Transactions on Internet Technology 11 (2011) 6:1–6:41.
[22] B. Videau, C. Touati, O. Richard, Toward an experiment engine for lightweight grids, in: Proceedings of the First International Conference on Networks for Grid Applications (GridNets 2007), GridNets '07, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2007, pp. 22:1–22:8.
[23] B. Videau, O. Richard, Expo : un moteur de conduite d'expériences pour plates-forme dédiées, in: Conférence Française en Systèmes d'Exploitation (CFSE).
[24] C. C. Ruiz Sanabria, O. Richard, B. Videau, I. Oleg, Managing Large Scale Experiments in Distributed Testbeds, in: Proceedings of the 11th IASTED International Conference, IASTED, ACTA Press, 2013, pp. 628–636.
[25] C. Ruiz, M. Alenxandru, O. Richard, T. Monteil, H. Aubert, Platform calibration for load balancing of large simulations: TLM case, in: CCGrid 2014 – The 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, Illinois, USA.
[26] C. Dwertmann, E. Mesut, G. Jourjon, M. Ott, T. Rakotoarivelo, I. Seskar, Mobile Experiments Made Easy with OMF/Orbit, in: K. Papagiannaki, L. Rizzo, N. Feamster, R. Teixeira (Eds.), SIGCOMM 2009, Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM, New York, NY, USA, 2009.
[27] T. Rakotoarivelo, M. Ott, G. Jourjon, I. Seskar, OMF: a control and management framework for networking testbeds, ACM SIGOPS Operating Systems Review 43 (2010) 54–59.
[28] G. Jourjon, T. Rakotoarivelo, M. Ott, From Learning to Researching - Ease the shift through testbeds, in: Internatinonal ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom), Springer-Verlag, Berlin, 2010, pp. 496–505.
[29] J. White, G. Jourjon, T. Rakotoarivelo, M. Ott, Measurement Architectures for Network Experiments with Disconnected Mobile Nodes, in: A. Gavras, N. Huu Thanh, J. Chase (Eds.), TridentCom 2010, 6th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, ICST, Springer-Verlag Berlin, Heidelberg, Germany, 2010, pp. 315–330.
[30] G. Jourjon, S. Kanhere, J. Yao, Impact of IREEL on CSE Lectures, in: the 16th Annual Conference on Innovation and Technology in Computer Science Education (ACM ITiCSE 2011), Germany, pp. 1–6.
[31] G. Jourjon, T. Rakotoarivelo, M. Ott, Why simulate when you can experience?, in: ACM Special Interest Group on Data Communications (ACM SIGCOMM) Education Workshop, Toronto, p. N/A.
[32] G. Jourjon, T. Rakotoarivelo, M. Ott, A Portal to Support Rigorous Experimental Methodology in Networking Research, in: T. Korakis, H. Li, P. Tran-Gia, H.-S. Park (Eds.), Testbeds and Research Infrastructure. Development of Networks and Communities, volume 90 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Berlin Heidelberg, 2012, pp. 223–238.
[33] M. Lacage, M. Ferrari, M. Hansen, T. Turletti, W. Dabbous, NEPI: Using Independent Simulators, Emulators, and Testbeds for Easy Experimentation, SIGOPS Oper. Syst. Rev. 43 (2010) 60–65.
[34] A. Quereilhac, M. Lacage, C. Freire, T. Turletti, W. Dabbous, NEPI: An integration framework for Network Experimentation, in: 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 1–5.
[35] C. Freire, A. Quereilhac, T. Turletti, W. Dabbous, Automated Deployment and Customization of Routing Overlays on Planetlab, in: T. Korakis, M. Zink, M. Ott (Eds.), Testbeds and Research Infrastructure. Development of

Networks and Communities, volume 44 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Berlin Heidelberg, 2012, pp. 240–255.

[36] A. Quereilhac, D. Camara, T. Turletti, W. Dabbous, Experimentation with large scale ICN multimedia services on the Internet made easy, IEEE COMSOC MMTC E-Letter 8 (2013) 10–12.

[37] T. Buchert, L. Nussbaum, Leveraging business workflows in distributed systems research for the orchestration of reproducible and scalable experiments, in: 9ème édition de la conférence Manifestation des Jeunes Chercheurs en Sciences et Technologies de l'Information et de la Communication (2012), Lille, France.

[38] T. Buchert, Orchestration d'expériences à l'aide de processus métier, in: ComPAS : Conférence d'informatique en Parallélisme, Architecture et Système., Grenoble, France.

[39] T. Buchert, L. Nussbaum, J. Gustedt, A workflow-inspired, modular and robust approach to experiments in distributed systems, in: CCGrid 2014 – The 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, Illinois, USA.

[40] M. Imbert, L. Pouilloux, J. Rouzaud-Cornabas, A. Lèbre, T. Hirofuchi, Using the EXECO toolbox to perform automatic and reproducible cloud experiments, in: 1st International Workshop on UsiNg and building ClOud Testbeds (UNICO, collocated with IEEE CloudCom 2013, Bristol, Royaume-Uni.

[41] R. Prodan, T. Fahringer, F. Franz, On using ZENTURIO for performance and parameter studies on cluster and Grid architectures, in: Proceedings of Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 185–192.

[42] D. Abramson, B. Bethwaite, C. Enticott, S. Garic, T. Peachey, Parameter Exploration in Science and Engineering Using Many-Task Computing, IEEE Transactions on Parallel and Distributed Systems 22 (2011) 960–973.

[43] R. Guillier, P. V.-B. Primet, A User-oriented Test Suite for Transport Protocols Comparison in Datagrid Context, in: Proceedings of the 23rd International Conference on Information Networking, ICOIN'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 265–269.

[44] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, Concurrency and Computation: Practice and Experience 18 (2006) 1039–1065.

[45] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, T. Oinn, Taverna: a tool for building and running workflows of services., Nucleic Acids Research 34 (2006) 729–732.

[46] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, H. T. Vo, VisTrails: visualization meets data management, in: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, SIGMOD '06, ACM, New York, NY, USA, 2006, pp. 745–747.

[47] J. Yu, R. Buyya, A Taxonomy of Scientific Workflow Systems for Grid Computing, SIGMOD Record 34 (2005) 44–49.

[48] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-Science: An overview of workflow system features and capabilities, Future Generation Computer Systems 25 (2009) 528–540.

[49] V. Curcin, M. Ghanem, Scientific workflow systems - can one size fit all?, in: Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International, pp. 1–9.

[50] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, D. Chalmers, The State of Peer-to-peer Simulators and Simulations, SIGCOMM Comput. Commun. Rev. 37 (2007) 95–98.

[51] H. Casanova, A. Legrand, M. Quinson, SimGrid: a Generic Framework for Large-Scale Distributed Experiments, in: Proceedings of the Tenth International Conference on Computer Modeling and Simulation, UKSIM '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 126–131.

[52] L. Leonini, E. Rivière, P. Felber, SPLAY: distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze), in: Proceedings of the 6th USENIX symposium on Networked systems design and implementation, NSDI'09, USENIX Association, Berkeley, CA, USA, 2009, pp. 185–198.

[53] W. Galuba, K. Aberer, Z. Despotovic, W. Kellerer, ProtoPeer: A P2P Toolkit Bridging the Gap Between Simulation and Live Deployement, in: Proceedings of the 2Nd International Conference on Simulation Tools and Techniques, Simutools '09, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2009, pp. 60:1–60:9.