HAL
open science

Managing Large Scale Experiments in Distributed Testbeds

Cristian Camilo Ruiz Sanabria, Olivier Richard, Brice Videau, Iegorov Oleg

To cite this version:

Cristian Camilo Ruiz Sanabria, Olivier Richard, Brice Videau, Iegorov Oleg. Managing Large Scale Experiments in Distributed Testbeds. [Research Report] RR-8106, INRIA. 2012. hal-00742582

HAL Id: hal-00742582

https://inria.hal.science/hal-00742582

Submitted on 16 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Managing Large Scale Experiments in Distributed Testbeds

Cristian Ruiz, Olivier Richard, Oleg Iegorov, Brice Videau

RESEARCH REPORT
N° 8106
October 2012
Project-Team MESCAL

ISSN 0249-6399 ISRN INRIA/RR--8106--FR+ENG

Managing Large Scale Experiments in Distributed Testbeds

Cristian Ruiz*†, Olivier Richard‡†, Oleg Iegorov†, Brice Videau§

Project-Team MESCAL

Research Report n° 8106 — October 2012 — 17 pages

Abstract: Performing experiments that involve a large amount of resources or a complex configuration, proves to be a hard task. We present Expo, which is a tool for conducting experiments on distributed platforms. First, the tool is described along with the concepts of resource and task sets, which abstracts away some of the complexity in the manage of resources and software execution. Next, the tool is compared with other similar solutions based on some qualitative criteria, scalability and expressiveness tests as well as the feedback coming from using dedicated testbeds. The report finishes with the evaluation of Expo scalability and some uses cases in Grid'5000 and PlanetLab. Our experience showed that Expo is a promising tool to help the user with two primary concerns: how to perform a large scale experiment efficiently and easily, together with its reproducilibity.

Key-words: Grid Computing, Distributed Application, Experiment conduction, Testbed, Experiment methodology.

* INRIA GRENOBLE
† LIG
‡ Université Joseph Fourier - Grenoble I
§ CNRS

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Conduite d'expériences de grande taille sur plates-formes d'expérimentation distribuées

Résumé : Conduire des expériences qui impliquent un grand nombre de ressources ou une configuration complexe, s'avère très difficile. Dans ce rapport, nous présentons Expo qui est un outil pour la conduite d'expériences sur les plates-formes distribuées. Ce rapport débute par la description de l'outil avec les concepts d'*ensemble de ressources* et d'*ensemble de tâches*, qui offrent des abstractions pour faciliter la gestion des tâches et des ressources. Ensuite, l'outil est comparé avec d'autres solutions basées sur des critères qualitatifs, et en utilisant des tests de scalabilité et d'expressivité ainsi que des retours après exécution sur des plates-formes d'expérimentation. Pour finir, la scalabilité d'Expo est evalué et quelques cas d'utilisation sur Grid'5000 et PlanetLab sont présentés. Notre expérience démontre qu'Expo peut aider les chercheurs sur deux points: comment conduire une expérience efficacement et facilement, et comment assurer sa reproductibilité.

Mots-clés : Expérimentation, Expériences à grande échelle, Plates-formes d'expérimentation, Méthodologie d'expérimentation

Contents

1 Introduction **4**

2 State of the Art **4**
2.1 Testbed Managers 5
2.2 Tools aimed to algorithm testing 5
2.3 Continous integration 6
2.4 Scientific Workflows 6
2.5 Parallel Executers 7

3 Expo **7**
3.1 Expo components 7
3.2 Abstractions 8
3.2.1 Resources 9
3.2.2 Tasks 9
3.2.3 Results 10

4 Some Uses cases **10**

5 Evaluation **11**
5.1 Evaluation of the experiment control systems 12

6 Conclusions and Future Works **14**

7 Acknowledgments **15**

1 Introduction

As the software to perform simulations have improved in recent years, there is still the need to test and evaluate the software in real distributed infrastructures. Moreover, the option of experimental evaluation of an algorithm has been encouraged as an approach complementary to the theoretical evaluation [9]. In order to addres limitations such as, software reconfiguration, lack of the control and access to monitoring systems testbeds were created [12]. A testbed is a platform for experimentation of large distributed applications. It is sometimes shielded from the instabilities of production environments, and allows users to test particular modules of their applications in an isolated fashion. Some examples of testbeds are: PlanetLab[18], Emulab[23], GENI [1], Grid5000[2] and ORBIT [15]. While these platforms offer more stability and control over resources than grids, controlling, deploying and running applications on them is still a hard task. That is the reason why some tools have been developed to cope with the problems encountered when researches try to perform experiments involving a large amount of resources or a complex configuration. The main aspects those tools help the user with, are: (1) the description of the experiment, (2) the control and access to the resources, (3) task orchestration, (4) software deployment, (5) monitoring and the collection of results. In more detail a number of tasks must be completed before an experiment can be actually started. These tasks include resource discovery and acquisition as well as the deployment of the necessary software. Once the application is launched, its execution must be controlled, and once it is finished all the output must be collected. As a result, most of the applications on the testbeds are run in an ad-hoc, application-specific manner. This method may match the current requirements of experiments, but fails with the scale, heterogeneity, and dynamism of highly distributed systems. We can classify the concerns when performing experiments in distributed infrastructures in two categories: how to deploy software and manage the nodes and how to provide a mechanism to describe and log the experiment activity in order to render this process more reproducible, which is the cornerstone of the scientific method [7]. Thereby the need of a generic tool to help users conduct their experiments in distributed environments is evident. In this paper we present Expo, already introduced in [21, 20], which is an experiment engine for distributed platforms. It helps users to conduct their experiments on any set of machines with the only requirement of remote shell access. We present its main advantages and compare it with others experiment control systems, as well as describing several uses cases and its scalability. The structure of this report is as follows: the next section shows the state of the art where related work is presented, regarding tools that an experimenter can use in order to conduct experiments in different testbeds. After in section 3 Expo is presented in depth with its principles and advantages, some cases of use are shown in Section 4 in two different testbeds. Results and comparison with other experiment tools are presented in Section 5 and finally section 6 presents the conclusions and future works.

2 State of the Art

There are many tools that researches can use in order to experiment with computational infrastructures. Here we recapitulate different complementary approaches passing from the testbeds managers, which are tools aimed to help the user with the experiment process, to other tools, which even though there are not directly related to the experimentation process, they share some similar concerns.

[1] http://www.geni.net

2.1 Testbed Managers

They are aimed at helping the user in the experimenting process, automating certain tasks. Sometimes they are tied to particular testbeds. Gush (GENI User Shell) - previously known as Plush[1] (PlanetLab User Shell) - is a widely used tool for application management on PlanetLab and GENI testbeds. It helps users to find desirable resources on the target platform, to prepare those resources with necessary software, to run the experiment, and to provide the maintenance while the experiment is running. With Gush users describe their experiments in a XML file, which consists of a set of building blocks. These blocks describe all the aspects of an experiment: software packages to install, resources to use and the main logic of an application. For partial tasks, such as resource discovery or remote software installation, Gush uses specific PlanetLab tools (SWORD, Stork, etc). As PlanetLab currently consists of 1133 nodes running at 544 sites all over the world, Gush's fault tolerance and scalability become extremely important features. Gush handles three types of failures: process failures, remote host failures, and controller failures (which is the responsible for managing the Gush client processes).

OMF (cOntorl and Management Framework) [19] is a testbed Control, Measurement and Management Framework. This framework is used in different wireless testbeds around the world and also in Planetlab. Its architecture versatility aimed at federation of testbeds, conceived mainly for the testing of network protocols and algorithms in wireless infrastructures. The OMF architecture consist of 3 logical planes: Control, Measurement, and Management. Those planes provide the user with tools to develop, orchestrate, instrument and collect results as well as the tools to interact with the testbed services.

It uses a comprehensive domain specific language based on Ruby to provide experiment-specific commands and statements. This allows an experimenter to write an experiment description, which details the resource requirements, their configuration, the definition of the events to capture and the respective actions to trigger when performing the experiment. They propose an approach in which an experiment will be similar to a meta package. It will depend on many other packages, containing the experiment description, applications, topologies and measurements.

The aim of the system in [6] is to extend Emulab capacities to support replayable research. They propose a new model for experimenting with testbeds which overcomes several problems encountered when interacting with Emulab. This model is composed mainly of two parts: templates and records. A template describes all the testbed-based environment, comprises every configuration file, source code, input files and any data generated or needed by the application. These templates are versioned, therefore they can suffer revision every time there is some change, which could be used for exploring different directions of research. A record is the persistent account of the activities and results that occurred within a run. This could be the data generated, all the raw intermediate data as well as the software, scripts, etc.

Additionally, there is a dynamic part that encompasses the activities of the experiments and how they are going to be scheduled when the experiment is executed as well as the actions triggered by the user or by special conditions. The user starts by describing a template, which defines the network topology and other experiment information. Then this template is commited to the system. The user next proceeds instanciating the template. When a template instance is created, testbed resources are allocated, configured, and booted. After the network and devices are up and running, the workbench automatically starts a run and starts any prescheduled activities withing that run.

2.2 Tools aimed to algorithm testing

Splay[13] is an integrated system, which covers the complete chain of distributed system design. It is a unique tool allowing to develop, deploy and maintain an application in a distributed

environment. Splay can be used on a classical testbed (PlanetLab, ModelNet, Grid5000), on a non-dedicated platform (such a network of idle workstations), in a normal cluster, and even on several testbeds simultaneously. Splay requires applications to be written in a Domain Specific Language based on Lua programming language. As Splay puts a particular emphasis on security, such applications are executed in sandboxed environments with minimal underlying operating system interaction. It uses a client/server architecture. The user can start his application on the server host or on some other machine, which can access the server. The server controls its execution and responds to failures. One of the most interesting features of Splay is its churn manager. It allows to reproduce a given live experiment with constant leaving and returning of participating nodes. The particular churn can be obtained from known analytical studies or publicly available churn scenarios.

Weevil[22] is a framework for automating experiments with distributed systems. It allows to generate application workloads based on a statistical model or on a specific system usage scenario programmed by the user. The latter feature makes Weevil a unique tool. After a specific workload is generated, the user can deploy his application on a predefined set of remote hosts, run it and collect the results. Thus, the work with Weevil consists of two stages: Workload generation and application deployment.

In [3] a tool to manage software in distributed infrastructures is presented. It is based on the Concept of Autonomous computing in order to make the administration of an infrastructure as a component architecture. The main idea is then to automatically create a representation based on fractal components of the real system, with two main parts: Application components and platform components. All expressed with a subset of UML diagram. It has already been used in the installation of a cluster software and the deployment of a TLM electromagnetic simulation code in a grid infrastructure[11].

2.3 Continous integration

Encompases different principles aiming at the rapid test and deploy of changes in software. It could be cited tools such as capistrano[2], which automate the process of server administration, mainly the deployment of web application using the Ruby and rails framework. NMI[16] is a framework to build and test software in a heterogeneous, multi-user, distributed computing environment. The principal aim is to offer to the user the continually testing of software changes. Because bugs in a software have to be fixed early in the development process. The user describe the process of building and testing along with its external software dependencies by using a lightweight declarative syntax. It takes advantage of a batch system (Condor) in order to deploy the software to be tested into the distributed infrastructure. Working with condor gives some capabilities to the system such as: matchmaking, fault tolerance, grid resource access, resource control, authentication and file transfer. It works along with a versioning system, so as to log the results and changes as well as performing the tracking of all input so that to ensure the repeatability and reproducibility of tests.

2.4 Scientific Workflows

These tools allow the experimenter to describe the complete flow of the experiment and the dependency between tasks and data generated. But the more important concept is the sharing of analysis and information through the composition and execution of workflows. One example in this category is Taverna [14].

[2] https://github.com/capistrano/capistrano/wiki

2.5 Parallel Executers

In this category we can cite projects like GridShell[5] and TakTuk[4], which help the user with the deployment of commands in large distributed infrastructures.

3 Expo

Expo[21, 20] [3] is a tool for conducting experiments on distributed platforms. It is written in Ruby and it has a modular structure, which allows to replace its components to add some new functionality. It is based on a client/server architecture. This architecture was designed having in mind that every time the lost of connection can occur, so that an asynchronous execution can be guaranteed. Communications between clients and servers are based on the SOAP protocol. Currently, Expo has been tested in Grid5000 and PlanetLab testbeds and uses its tools to perform subtasks. The core functionality of Expo consists in taking commands from the user and execute them efficiently on a large set of heterogeneous distributed resources. This efficiency is achieved thanks to parallel execution using TakTuk[4], which spread itself and form a tree, improving the scalability and enabling the exploitation of the hierarchy structure of the platform.

Expo users describe their experiments using Domain Specific Language (DSL) based on Ruby. As a result of the feedback obtained from Grid5000, we wanted to focus on the deployment and efficient execution of the experiment. That is why Expo offers the possibility of executing any software, as long as such software can be executed on the target machine. It interacts with the services of the Grid5000 testbed such as its API [4] in order to provide an easy interaction with Kadeploy [8], which allows the user to deploy complete operating systems. It is not in any way tied to a specific platform, because of its modular architecture new modules can be added in order to interact with other testbeds or platforms. This was proved by developing the module for interacting with PlanetLab, which did not require any change in the subjacent components.

3.1 Expo components

Expo is the result of the interaction of different components that offer several functionalities. This interaction is shown in Figure 1. It was developed having in mind a modular architecture, each one of the component is described below:

- **Command Executer:** It gives verbosity to the commands, getting all the command status and giving a low level control execution.

- **Experiment Controller:** This part keeps track of the whole experiment run, controlling all the information related to an experiment. It logs all the commands executed, storing information such as: time of execution, output, errors etc. It enables to unify the tasks and resources into a logical unit: the experiment, making possible the addition of tasks or resources. The client part enables to save all the information into a structured data format for a postmortem study.

- **Reservation System:** The request of resources is perfom by this component, using the Command Executer and additionally, it associates all the reservation information to the experiment.

[3] http://expo.gforge.inria.fr/
[4] http://www.grid5000.fr/mediawiki/index.php/API

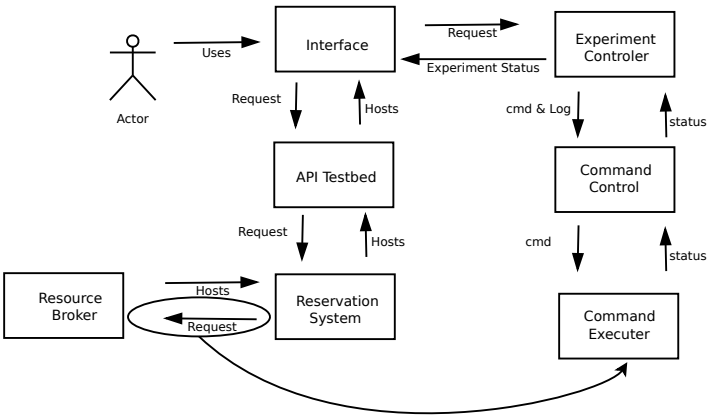


Figure 1: Expo Component Diagram

- **API Testbed:** It interacts with the services provided by the testbed. Information about the platform can be fetched by this component, as well as the acquisition of resources. If the platform uses a lease-based approach to manage the resources, it interacts with the reservation system.

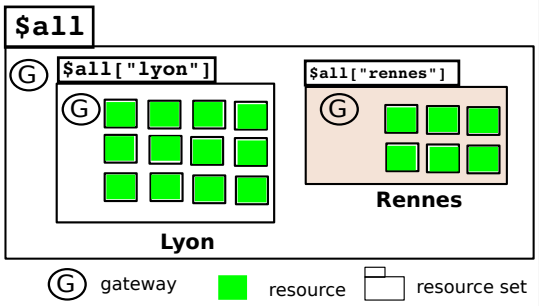
- **Interface:** This part could be seen as the set of interacting methods. An interactive mode driven by the following reasons: (1) an important part of the experiments are interactive, (2) the writing of an experiment description file is a trial-and-error process which involve using different parameters, configurations and flows of control, (3) "*An interctive environment lets scientists look at data, test new ideas, combine algorithmic approaches, and evaluate their outcome directly*"[17]. This approach is already used by different scientific environments based on Python such as: IPython and Scipy[10]. A Stand alone mode is also available, which execute the experiment description file without any user intervention.

  synchronous and asynchronous execution of commands. It allows the user to access the information about the finished commands or those that are running thanks to a unique identifier. The information available are: Command status, start and end dates of the command, return status, *stdout* and *stderr*.

Later, it is shown the interface to all these components by presenting the language for describing the experiment. The features that control the flow of execution and the commands provided by Expo are described in [20].

3.2 Abstractions

In this subsection the abstractions are presented in depth, which is one of the more important features of Expo and some operations that have been added in order to ease and improve the



(a) Resource hierarchy.

```
require 'g5k_api'

g5k_init(
        :site => ["lyon","rennes"],
        :resources => ["nodes=12","nodes=6"],
        :walltime => 7200
          )

g5k_run

## All The resource reserved
$all
$all.gateway
## New resource set
##with only resources of lyon cluster
lyon_nodes=$all["lyon"]
lyon_nodes.gateway
```

(b) Expo Experiment description File.

Figure 2: Example of exploiting the hierarchy in Grid5000 using Expo. Here is presented grapically the concept of resource Set and how it looks like in the code of the experiment description file.

flexibility of the experiment description. These abstractions are implemented as objects in Ruby and have some operations, properties and iterators. They resemble other existing objects in Ruby.

3.2.1 Resources

Expo introduces the notion of ResourceSet. These ResourceSet add resources into a logical unit and associate properties to them. For instance, we can gather together the nodes from a same cluster associating them the same frontend, as well as the same physical properties if the cluster is homogeneous. This information is actually used in order to perform the efficient deployment of commands. These sets of resources could suffer diverse transformations using some operations and properties summarized in Table 1.

Figure 2 shows how the resourceSet object can be used to exploit the hierarchy of the infrastructure in Grid5000. We can divide the resources belonging to the same site as well as separate them per cluster. This can also be applied for the Planetlab testbed e.g. the resourceSet can have information about the location of the resources for the same country or site. In other cases it can be used to define complex configurations as in the case we need to deploy an infrastructure where different nodes have different roles. and their interactions.

3.2.2 Tasks

A task associate a command with a set of resources. Therefore, different mappings between commands and resources can be expressed and easily managed. It can either encapsulates the parallel or the sequential execution of commands. Also it is possible to create a set of tasks, each one with different characteristics and resources. On those set of tasks, it is possible to control the way tasks are executed such as synchronously or asynchronously.

Operation	Description
[]	Subset of the resource Set, nodes[1..4] creates a subset with 4 resources.
to_resources	Returns an array composed of resource objects.
length()	Returns the number of resources in the ResourceSet.
each_slice_power2	Iterates over subsets composed of power of 2 resources.
resource_file	Creates a file with the resources' hostnames.
delete_if	Deletes resource if the condition is true.
push	Adds more resources to the Set.

Table 1: Resource Set Operations

3.2.3 Results

This class encapsulates the results, allowing the user to the postmortem treatment of the execution of the experiment. Obviously the gathering of this information has a cost, but it does not influence the self execution of the experiment. Among its features, it wraps up in ruby objects different functions normally used in an experiment, in order, for example, to measure time or getting some statistics. Making easier for the user the treatment of his/her experiment.

4 Some Uses cases

The aim of this section is to show how the experiments are described with Expo. The different functionalities that provide the user with the flexibility for describing his/her experiment. Additionally, this section shows the interaction with two different testbeds, Grid5000 and PlanetLab. In Figure 3 two examples of experiment description files are shown. One experiment description is an experiment performed in Grid5000, which consisted in the execution of an electromagnetic simulation over different sites using one node per site. We deployed an OS image with all the software already installed through the use of Grid5000 API that interacts with Kadeploy. The specification of the corresponding image to deploy is indicated as a parameter in the function that request the resources, which it is shown in the first lines of the file. It is important to remark that the specification of the resources to use has a lot of parameters, because we needed this especial configuration but most of the parameters shown can be ignored and the default values will be used. Moreover, in the file we can see some Expo operators to ease the procedure of execution of a program using MPI such as generating a secure ssh communication and generating the correct hostfile. Next all the infiniban interfaces are deactivated, which is easily expressed with Expo. The application is executed and we got the results of the execution in two variables that we can treat with different operators and methods already defined in them. Finally we gather the files generated by the execution from the nodes with just one command. The other experiment performed over Planetlab consisted in executing the Linux command "hostname" in all the nodes of the slice, and to count how many of them reply. This information is put in a file that can be used to plot the availability of the nodes in the slice. It is a simple example that show how an experiment is carried out in Planetlab using Expo. More examples can be found on the Expo website [5]

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

```
require 'g5k_api'

base_url = "http://public.grenoble.grid5000.fr/~ruizsan/"
environment = "tlm_simulation.env"
g5k_init(
        :site => ["nancy","rennes","lille",
      "grenoble","sophia"],
        :resources => ["nodes=1"],
        :environment => {base_url+environment => 5},
        :walltime => 3600,
        :deployment_max_attempts => 3,
        :name => "TLM_code"
  )
g5k_run


$all.gen_keys

copy $all.node_file, $all.first ,:path=> "/root/nodes.deployed"

### deactivating the ib0 interface
$all.each { |node|
                task node, "/sbin/ifconfig ib0 down"
}
### Execution of the simmulation with MPI
id,res = task $all.first ,"./exec_tlm 1 369 192 510 250 1 sc"

get_results($all,"/TLMME_multimode/profile.*","~/profiles")
```

(a) TLM experiment description.

```
require 'expo_planetlab'
## getting resources form planetlab Slice
get_resources
task_mon=Task::new("hostname",$all,"Monitoring")
File.open("Planetlab_avail.txt",'w+'){|f|
  res=nil
  f.puts "Date  Time  Num_Res"
  240.times{
    data_me=Time::now.to_i
    id, res = task_mon.execute
    time=res.total_duration
    f.puts "#{data_me}  #{time}  #{res.length}"
    f.flush
    sleep(60)
    }
}
```

(b) PlanetLab experiment.

Figure 3: This show two examples of experiment description files used in different testbeds, Grid5000 and Planetlab. It is important to notice that we just need to load another module in order change the platform.

5 Evaluation

As our platform of study is Grid5000, we took most of the feedback from there and the use cases. Some percentage of the experiments use jobs, which run interactive, some are batch and some need the deployment of an image as seen in Table 2. Some of the difficulties that the user has to deal with are:

- Managing heterogeneous clusters, varying the number of sites, clusters and computational nodes.

Type	**Number**
Testing software that already exist	52 %
Deployment of environment	33 %
Batch mode	47 %
Interactive mode	53 %

Table 2: Experiments in Grid5000

	Gush	Splay	Weevil	OMF	Expo	Workbench
Workload Generation	-	-	+	-	-	-
Resource Discovery	+	+	-	+	+	+
Fault Injection		-	+	-	-	-
Software Deployment	+	-	+	+	+	+
Fault Tolerance	+	+	+	+	-	-
Expresivennes	-	+	-	+	+	+
Instrumentation	-	+	-	+	-	-
Ligthweigth approach	+	-	-	-	+	-
Versioning system	-	-	-	+	-	+
Platform	any	any	any	Orbit	any	Emulab

Table 3: Tools comparison

- Specifying several steps for the execution of commands.
- Choosing and keeping track of the sites used.
- Deploying software.
- Easy control of nodes.
- Running experiments at very large scale with more than 1000 nodes.

We used those difficulties as criteria for comparing the different experiment control systems found and to justify our design choices in the development of Expo.

5.1 Evaluation of the experiment control systems

In section 2, we presented some of the tools that an experimenter can take advantage from. In this section, we focus on the tools that were conceived to help the user with the experiment cycle in distributed systems. The evaluation start by comparing those tools. In Table 3, we show the different criteria used to compare them. We can see that some of them favor the reproducibility of experiments like the capacity of workload generation, fault injection and the use of a versioning system. Others are platform dependent and we can see also some important characteristic such as the ease of use. The aim of this evaluation is to place our tool regarding all the existing experimental frameworks as well as looking for characteristic that are worth to be taken into account for Expo. Weevil regardless of its capacity of generating a workload and helping the user with the creation of scripts, it uses for describing the experiment a language used to create configuration files for complex software, which few users are aware of. OMF installation requires several nodes to perform a simple installation and it is more oriented to the

```
  <?xml version="1.0" encoding="utf-8"?>
<gush>
  <project name="Testing overhead">
    <component name="Cluster1">
        <rspec>
            <num_hosts>20</num_hosts>
        </rspec>
        <resources>
          <resource type="ssh" group="local"/>
        </resources>
    </component>

         <experiment name="simple">
            <execution>
               <component_block name="cb1">
                  <component name="Cluster1"/>
                  <process_block name="p2">
                     <process name="test">
      <path>hostname</path>
                           <cmdline>
                             <arg></arg>
                           </cmdline>
                     </process>
                  </process_block>
               </component_block>
            </execution>
         </experiment>
  </project>
</gush>
```

(a) Gush Experiment Description.

```
require 'g5k_api'
require 'benchmark'

g5k_init(
        :site => ["nancy","sophia"],
        :resources => ["nodes=200","nodes=100"],
        :walltime => 2000,
        :name => "Big_experiment"
)

g5k_run
# run the reservation

sizes=[10,50,100,200,300]

puts "nb_nodes  time"
### iterates over the differents sets to try
sizes.each{ |n|

         nodes=$all.uniq[0..(n-1)]
         task_mon=Task::new("hostname",nodes,"Testing")
    res=0
         20.times{
             id, res = task_mon.execute
             puts "#{res.length} #{res.total_duration}"
         }
}
```

(b) Expo Experiment Description.

Figure 4: Comparison between Experiment description files: These files were used in the evaluation of the scalability of the two tools. It should be noticed here that the experiment description for Gush has to be changed every time we need to change the number of nodes to try with. Also that Gush needs a file for the resource description that is not shown.

network experimentation. Unlike Expo which requires only the presence of Ruby and Perl and some modules in order to work. However, the language provided by OMF for describing the experiments offers a big flexibility and uses the same approach as Expo. Splay was conceived to test algorithms and needs everything to be implemented in the Lua language, which makes it difficult to use it for testing and deploying existing software. Workbench is completely bounded to Emulab and does not support the manage of different testbeds. In contrast to almost all the tools, Gush shares many features with Expo as the ease of installation and the capacity of being adaptable to different testbeds and that is the reason why we evaluate both in more depth.

The evaluation consisted in the expressiveness of the language, the performance and scalability of the command execution. The comparison between both tools was done by carrying out an experiment, which involved a large amount of nodes. We defined an experiment that consisted in executing a command in a set of resources and measuring the time elapsed, while varying the number of nodes. Therefore, we compare the time to execute the commands and the flexibility in the description of the experiment.

Figure 4 shows the descriptions of the experiment used for Gush and Expo. We can notice looking at the experiment description that for Gush either we have to change the file for each experiment so as to try different number of resources, or we can create a long description file with all the possibilities we want to try. This is not the case for Expo, which uses Ruby and provides a programmatically approach for describing the experiment allowing it to be flexible enough to adapt to the normal activities or changes when we perform an experiment.

In Figure 5 the scalability of the mechanism for the execution of commands was evaluated. In this figure, we can see that Expo outperforms Gush due to the use of TakTuk parallel executer, also that Expo presents less variability in the time to execute the experiment, which is important to the reproducibility. It was noticed as well that when we tried to execute an experiment with more than 400 nodes, problems arise trying to perform it with Gush.

6 Conclusions and Future Works

Experimentation in computer science and specially in distributed infrastructures has seen the emergence of different experiment control systems and from those we can draw as a conclusion that most of the tools distinguish almost the same phases in the experimenting process. The parts of the experiment process that a tool must control and help the user with are mainly three: (i) the control, (ii) the supervision and (iii) the management of the experiment. The first part comprises the description of the experiment, the capture of data, the definition of the source of data and how to get it after the experiment has finished as well as the flow of control of the experiment. This is an important step for the reproducibility of the experiment. Second, the experiment supervision, which means the monitoring of the experiment. As a last phase the experiment management, i.e. the interaction with the platform, how to take advantage of the services provided by the infrastructure in order to carry out the experiment.

Expo offers a way to describe the experiment by using a programming language providing a lot of flexibility and more importantly the abstractions that allows the user to express complex configurations. We had special attention in automating the typical tasks done when an experiment is performed. Because we think that automating the experimentation process is the way to go, being one of steps that will lead to the experiment reproducibility. Furthermore it is important to encourage the culture of experiment reproducibility, which is acknowledged to be a shortcoming in computer experimentation.

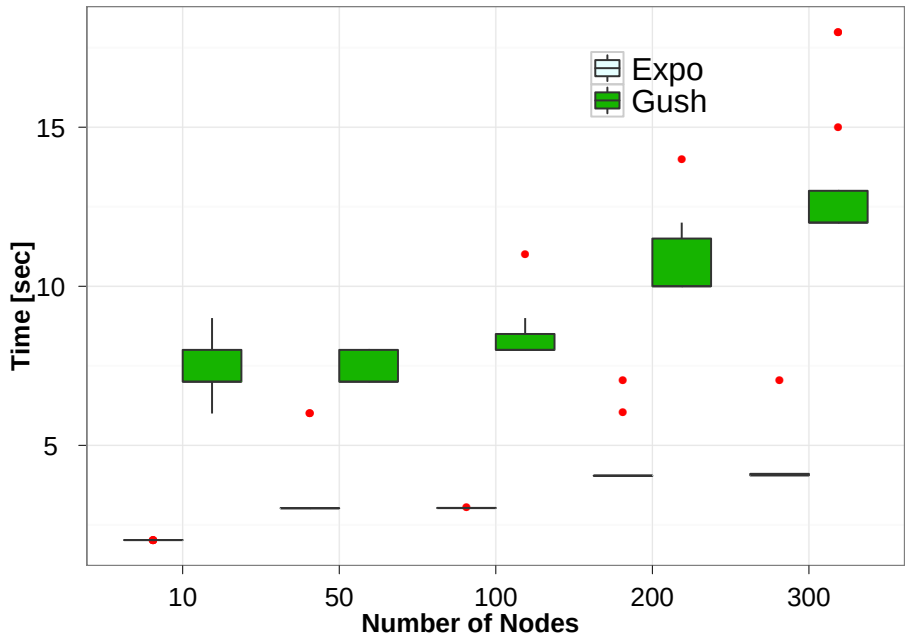


Figure 5: Evaluation of the scalability of Gush and Expo when executing a command in a large set of resources. The red points represent outliers.

The use of experiment tools will save user time, which can be spent in improving the software itself, it will save costs and allow others to reproduce the results more easily. It is important to integrate some features to Expo for the sake of reproducibility. We need to improve the part of the system that logs the experiment run in order to have detailed and easy to treat information that would enable a possible replay of the experiment. Incorporate some mechanisms to monitor and to generate a workload and more importantly to deal with fails.

7 Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] Jeannie Albrecht, Ryan Braud, Darren Dao, Nikolay Topilski, Christopher Tuttle, Alex C. Snoeren, and Amin 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, pages 15:1–15:19, Berkeley, CA, USA, 2007. USENIX Association.

[2] RaphaÃ«l Bolze, Franck Cappello, Eddy Caron, Michel DaydÃ©, Frederic Desprez, Emmanuel Jeannot, Yvon JÃ©gou, StÃ©phane Lanteri, Julien Leduc, Noredine Melab, Guil-

laume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and TouchÃ© Irena. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, nov 2006.

[3] Laurent Broto, Daniel Hagimont, Patricia Stolf, Noel Depalma, and Suzy Temate. Autonomic management policy specification in tune. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 1658–1663, New York, NY, USA, 2008. ACM.

[4] Benoit Claudel, Guillaume Huard, and Olivier Richard. Taktuk, adaptive deployment of remote executions. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 91–100, New York, NY, USA, 2009. ACM.

[5] T. Minyard E. Walker and J. Boisseau. Gridshell: A login shell for orchestrating and coordinating applications in a grid enabled environment. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, HPDC-13*, june. 2004.

[6] Eric Eide, Leigh Stoller, and Jay Lepreau. An experimentation workbench for replayable networking research. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, NSDI'07, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.

[7] Dror G. Feitelson. Experimental computer science: The need for a cultural change, 2006.

[8] Yiannis Georgiou, Julien Leduc, Brice Videau, Johann Peyrard, and Olivier Richard. A Tool for Environment Deployment in Clusters and light Grids. In *IPDPS*, Rhodes Islands, États-Unis, 2006. IEEE.

[9] David Johnson. A theoretician's guide to the experimental analysis of algorithms, 1996.

[10] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[11] Fadi KHALIL. Multi-scale modeling: from electromagnetism to grid, 2009.

[12] Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quetier, and Olivier Richard. Grid'5000: a large scale and highly reconfigurable grid experimental testbed.

[13] Lorenzo Leonini, Étienne Rivière, and Pascal 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, pages 185–198, Berkeley, CA, USA, 2009. USENIX Association.

[14] Thomas Oinn, Mark Greenwood, Matthew Addis, Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Christopher Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, August 2006.

[15] Maximilian Ott, Ivan Seskar, Robert Siraccusa, and Manpreet Singh. Orbit testbed software architecture: Supporting experiments as a service. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and*

COMmunities, TRIDENTCOM '05, pages 136–145, Washington, DC, USA, 2005. IEEE Computer Society.

[16] Andrew Pavlo, Peter Couvares, Rebekah Gietzel, Anatoly Karp, Ian D. Alderman, Miron Livny, and Charles Bacon. The nmi build & test laboratory: continuous integration framework for distributed computing software. In *Proceedings of the 20th conference on Large Installation System Administration*, LISA '06, pages 21–21, Berkeley, CA, USA, 2006. USENIX Association.

[17] Fernando Pérez and Brian E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007.

[18] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building planetlab. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 351–366, Berkeley, CA, USA, 2006. USENIX Association.

[19] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. Omf: a control and management framework for networking testbeds. *SIGOPS Oper. Syst. Rev.*, 43(4):54–59, January 2010.

[20] B. Videau and O. Richard. Expo : un moteur de conduite d'experiences pour plates-forme dedies. In *Conference Fransaise en Systemes d'Exploitation (CFSE)*, 2008.

[21] Brice Videau, Corinne Touati, and Olivier Richard. Toward an experiment engine for lightweight grids. In *MetroGrid workshop : Metrology for Grid Networks*. ACM publishing, October 2007.

[22] Yanyan Wang, Matthew J. Rutherford, Antonio Carzaniga, and Alexander L. Wolf. Automating experimentation on distributed testbeds. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 164–173, New York, NY, USA, 2005. ACM.

[23] 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. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, December 2002.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399