ABSTRACT
Performing experiments that involve a large amount of resources or a complex configuration proves to be a hard task. In this paper 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 experiment conduction. 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 paper finishes with the evaluation of Expo scalability and some use cases on Grid5000 and PlanetLab testbeds. Our experience showed that Expo is a promising tool to help the user with two primary concerns: performing a large scale experiment efficiently and easily, together with its reproducibility.

KEY WORDS
Grid Computing, Distributed Application, Experiment conduction, Testbed, Experiment methodology.

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[1]. In order to address limitations such as, software reconfiguration, lack of control and monitoring systems, testbeds were created[2]. A testbed is a platform for experimentation with 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[3], Emulab[4], GENI 1, Grid5000[5] and ORBIT [6]. Although these platforms offer more stability and control over resources, it is still a hard task controlling, deploying and running applications on them. That is the reason why some tools have been developed to cope with the problems encountered when researchers 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) description of the experiment, (2) control and access to the resources, (3) task orchestration, (4) software deployment, (5) monitoring and 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 deployment of the necessary software. Once the application is launched, its execution must be controlled, and as soon as it finishes 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: a) deployment of software and manage of a distributed set of machines, b) experiment reproducibility. The latter 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 [8, 9], 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 other experiment control systems as well as describe several use cases and its scalability. The structure of this paper 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 features and advantages, some cases of use are shown in Section 4 in two different testbeds. Results and comparisons with other experiment tools are presented in Section 5 and finally section 6 presents the conclusions and future works.

1http://www.geni.net
2 State of the Art

There exists variety of tools that researchers 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, that even though there are not directly related to the experimentation process, they share some similar concerns.

2.1 Testbed Managers

They are aimed at helping users in the experimenting process, automating certain tasks. Sometimes they are tied to particular testbeds. Gush (GENI User Shell) - previously known as Plush[10] (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.

OMF [11] is a framework used in different wireless testbeds around the world and also in Planetlab. Its architecture versatility aimed at federation of testbeds, conceived mainly for testing network protocols and algorithms in wireless infrastructures. The OMF architecture consists 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. This approach will provide support for systematic experiment life-cycle and reproducibility.

Workbench [12] extends Emulab capacities to support replayable research. It proposes a new model for experimenting with testbeds that 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 environments, comprises every configuration file, source code, input files and any data generated or needed by the application. These templates are versioned, and as such 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. It 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: a) the activities of the experiment, b) the scheduling of these activities when the experiment is executed, c) 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 committed to the system. The user next proceeds instantiating the template. When a template instance is created, testbed resources are allocated, configured, and booted. Once the network and devices are up and running, the workbench automatically initiate 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 (e.g., 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. A client/server architecture is used in Splay. 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[14] 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. Providing all the specifications, Weevil automatically generates deployment scripts and run the application.
2.3 Autonomic management

Tune[15] a tool to manage software in distributed infrastructures. 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[16].

2.4 Continuous integration

It encompasses different principles aiming at the rapid test and deploy of changes in software. It could be cited tools such as capistrano2, which automate the process of server administration, mainly the deployment of web application using the Ruby and rails framework. NMI[17] 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 give 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 reproducible of tests. It is used in the development of Globus toolkit and Condor Batch systems.

2.5 Scientific workflows

Several scientific workflow management systems have been developed for computational science. 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. Examples in this category are Taverna [18] and VisTrails [19].

2.6 Parallel executer

In this category we can cite projects like GridShell[20] and TakTuk[21], which help the user with the deployment of parallel remote executions in large distributed infrastructures.

3 Expo

Expo[8, 9]3 is a tool for conducting experiments on distributed platforms. It is written in Ruby and it has a modular structure, which enables its components to add some new functionality. Expo is based on a client/server architecture. This architecture was designed having in mind that a loss of connection can occur any time. An asynchronous execution can be thus guaranteed. Communications between clients and servers are based on the SOAP protocol. Currently, Expo has been tested in Grid5000 and PlanetLab testbeds where it uses all the services provided by those testbeds to perform sub-tasks. 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[21], 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 (e.g its API4) in order to provide an easy interaction with Kadeploy [22], which allows the user to deploy complete operating systems. Expo 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. Expo’s components are described below: 

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

- **Experiment Controller**: 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

---

2https://github.com/capistrano/capistrano/wiki
3http://expo.gforge.inria.fr/
4http://www.grid5000.fr/mediawiki/index.php/API
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**: performs the request of resources by using the Command Executer and associate all the reservation information to the experiment.

- **API Testbed**: interacts with the services provided by the testbed. It could be used to get information about the platform, as well as to get the 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 involves using different parameters, configurations and flows of control, (3) “An interactive environment lets scientists look at data, test new ideas, combine algorithmic approaches, and evaluate their outcome directly”[23]. This approach is already used by different scientific environments based on Python such as: IPython and Scipy[24]. A Standalone mode is also available, which execute the experiment description file without any user intervention.

- **Command Control**: provides the control of the commands execution by using a scheduler, enabling the 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 [9].

3.2 Abstractions

In this subsection the abstractions are presented in depth (which is one of the more important features of Expo) as well as some operations that have been added in order to ease and improve the 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 the same cluster associating to 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.

3.2.2 Tasks

A task associates a command with a set of resources. Therefore, different mappings between commands and resources can be expressed and easily managed. It can either encapsulate 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. It is possible then to control the way tasks are executed (synchronously or asynchronously).

3.2.3 Results

This class encapsulates the results, allowing the user to perform a postmortem treatment of the experiment execution. Obviously the gathering of this information has a cost, but
Figure 2: Example of exploiting the hierarchy in Grid5000 using Expo. Here is presented how the resources are really distributed and how it looks like in the code of the description file.

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

Table 1: Resource Set Operations

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 (e.g. time measurement, retrieval of statistics, etc.). It thus makes easier for the user the handling of his/her experiment.

4 Some Uses Cases

The aim of this section is to show how the experiments are described with Expo using 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 Listing 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.

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 the correct hostfile. Next, all the infiniband interfaces are deactivated, which is easily expressed with Expo. We execute the application and we get the results of the execution in two variables that we can treat with different operators and methods defined in the class Result. Finally, we gather the files generated during the execution from the nodes with just one command. The other experiment executed over Planetlab consisted in executing the Linux command “hostname” on all the nodes of the slice, and counting 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 Evaluation

As our platform of study is Grid5000, we took most of the feedback and use cases from this platform. Some of the difficulties that the user has to deal with are:

- Managing heterogeneous clusters, varying the number of sites, clusters and computational nodes.
- Specifying several steps for the execution of commands.
- Choosing and keeping track of the sites used.
- Deploying software.
- Control of nodes.
- Running experiments at very large scale with more than 1000 nodes.

6http://expo.gforge.inria.fr/
### 5.1 Evaluation of the experiment control systems

In section 2, we presented some of the tools that an experimenter can take advantage of. 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 2, 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. Other tools are platform dependent and we can see as well some important characteristic such as the usability. 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, uses a specific language to describe experiments and create configuration files, which few users are aware of. OMF installation requires several nodes to perform a simple installation and it is more oriented to the 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 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 to adapt to different testbeds. Both Gush and Expo are targeted for performing general experiments in distributed infrastructures. That is the reason why we evaluate both in more depth.

The evaluation consisted in the expressiveness of the language, as well as 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. Listing 4 shows the descriptions of the experiment used for Gush and Expo. We can note, looking at the experiment description, that for Gush we have to either change the file for each experiment so as to try different experiment used for Gush and Expo. We can note, looking at the experiment description, that for Gush we have to either change the file for each experiment so as to try different experiment used for Gush and Expo. We can note, looking at the experiment description, that for Gush we have to either change the file for each experiment so as to try different experiment used for Gush and Expo. We can note, looking at the experiment description, that for Gush we have to either change the file for each experiment so as to try different experiment used for Gush and Expo. We can note, looking at the experiment description, that for Gush we have to either change the file for each experiment so as to try different

(a) TLM experiment description.

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.

Table 2: Tools comparison

<table>
<thead>
<tr>
<th></th>
<th>Gush</th>
<th>Splay</th>
<th>Weevil</th>
<th>OMF</th>
<th>Expo</th>
<th>Workbench</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload Generation</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Resource Discovery</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Fault Injection</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Software Deployment</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Expressiveness</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Lightweight approach</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Versioning system</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Platform</td>
<td>any</td>
<td>any</td>
<td>any</td>
<td>Orbit</td>
<td>any</td>
<td>Emulab</td>
</tr>
</tbody>
</table>
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 programatically 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 mechanism for the execution of commands was evaluated in order to see the scalability. 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 Conclusion and Future Works

Experimentation in computer science and specially in distributed infrastructures has seen the emergence of different experiment control systems. From this fact we can draw as a conclusion that most of the tools distinguish almost the same phases in the experimenting process. There are three main parts of the experiment process that a tool must control and help the user with: (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. The last phase is the experiment management, which is the interaction with the platform, and mainly consist in taking 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 allow the user to express complex configurations. We put special attention at
automating the typical tasks done when an experiment is performed. Because we think that automating the experiment 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.

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. It will be also important to incorporate some mechanisms to monitor and to generate a workload, and more importantly, to deal with fails.

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


