Introducing PRECIP: An API for Managing Repeatable Experiments in the Cloud

Sepideh Azarnoosh, Mats Rynge, Gideon Juve, Ewa Deelman
University of Southern California
Information Sciences Institute
Marina del Rey, U.S.A.
{azarnoos, rynge, gideon, deelman}@isi.edu

Michal Nieć, Maciej Malawski
AGH University of Science and Technology
Department of Computer Science
Krakow, Poland
michalniec@gmail.com
malawski@agh.edu.pl

Rafael Ferreira da Silva
Univ. Lyon, CNRS, CREATIS
Villeurbanne, France
rafael.silva@creatis.insa-lyon.fr

***Abstract*—Cloud computing with its on-demand access to resources has emerged as a tool used by researchers from a wide range of domains to run computer-based experiments. In this paper we introduce a flexible experiment management API, written in Python, that simplifies and formalizes the execution of scientific experiments on cloud infrastructures. We describe the features and functionality of PRECIP (Pegasus Repeatable Experiments for the Cloud in Python), and how PRECIP can be used to set up experiments on academic clouds such as OpenStack Eucalyptus, Nimbus, and commercial clouds such as Amazon EC2.**

Keywords—Cloud; Experiment Management; Cloud Provisioning, OpenStack; Eucalyptus; Nimbus; Amazon EC2

I. INTRODUCTION

A. Motivation

The foundations of modern science are to observe a phenomenon, formulate a hypothesis, develop an experiment, and analyze the results. In many scientific fields, experimental conditions are carefully controlled, and experiments are required to be described in enough detail to be reproducible by other researchers. Computer science experiments are often conducted with a less formal approach, using computer apparatus and configuration that is not easily described or reproducible outside the researcher's laboratory. In addition, researchers might not have sufficient local computer resources for the experiments required to test their hypothesis. One solution to these problems is to use cloud computing infrastructures, which are becoming increasingly popular due to the configurability and flexibility they provide [1]. A popular experiment infrastructure is FutureGrid [2][3], a NSF-funded cloud test bed designed for use in academic research. It currently has 303 active projects [4], the majority of which are computer science experiments. These projects span many domains, including computer science (48%), education (14%), life science (10%), and others. While setting up an experiment in a cloud environment can be straightforward, important components of the scientific method, such as the recording of the experiment and the ability to later reproduce the experiment, are frequently left unresolved.

Keeping good records of an experiment is critical in order to present and defend findings. In traditional science experiments, a laboratory notebook is maintained as an official record of the experiment. These notebooks enable the scientist to go back and reproduce the experiment, re-examine the data, and compare the procedure and results with other experiments. Similar to a laboratory notebook, computer science experiments should result in a similar recording that captures what steps were taken, at what point in time. For example, a verbose log with timestamps can fulfill the recording requirement.

Researchers have to be able to reproduce experiments for several reasons; to iterate on the scientific method, to evaluate the process and results of other researchers' experiments, and to share their own experiments with other researchers for collaborative or peer reviewing purposes. In cloud-based computer science experiments, reproducibility can be aided by the cloud infrastructure by storing virtual machines used for the experiment, but with the current tools the steps of the experiment are not usually captured automatically. Just as for the manual experiments, the scientist can maintain a laboratory notebook with the steps, but this is not scalable for computer science experiments with a large number of resources or steps. Also, it is common for computer science experiments to require precise timing of the steps, which can be difficult to do manually. To enable reproducibility and automatic recording, new innovative tools are needed. These tools should guide the researcher to provide self-contained, fully described, reproducible, and peer reviewable cloud experiments.

While formalizing computer science cloud experiments for the scientific method was the main motivation for this work, there are additional challenges a scientist faces when setting up experiments on cloud infrastructures. One such challenge is API fragmentation. FutureGrid provides a number of different cloud management systems, including: OpenStack [5], Eucalyptus [6], and Nimbus [7]. Even though all of these provide an Amazon EC2 interface, they all differ in small and subtle ways. These differences mean that the researcher could spend more time learning how to execute their experiment on the various infrastructures than they do running the experiment itself. The fragmentation also adds more complexity on running experiments across infrastructures. Therefore, a secondary goal of our work is to provide a thin abstraction layer on top of the available cloud interfaces.

B. Contribution

Our main contribution is an experiment management API called PRECIP (Pegasus Repeatable Experiments for the Cloud In Python). Python was chosen because of its ease of use and ubiquity in academic computing. Only a minimal amount of Python knowledge is required to use PRECIP. Although Python is used to orchestrate the experiment, the experiment itself is not required to use Python. In summary, PRECIP provides:

1. A thin abstraction API to support cloud interoperability. The researcher only needs to learn one API in order to run experiments across multiple clouds, or to move an experiment from one cloud to another.
2. Reproducibility through scripting. The researcher can easily run the same experiment over and over again with reproducible results.
3. Automatic logging. PRECIP steps are time stamped and logged.
4. Powerful instance tagging features to simplify the management of experiment resources.
5. Basic handling of provisioning, SSH keys, and security groups in a fault tolerant manner.

The rest of this paper is organized as follows. Section 2 presents an overview of targeted experiments requirements. Section 3 describes PRECIP's design and its main functionalities. Fault tolerance is discussed in section 4. A sample experiment is shown in Section 5. Section 6 is about related work, and Section 7 concludes the paper.

II. Targeted Experiments Requirements

In order to develop PRECIP into a widely usable product, we examined registered FutureGrid projects, and loosely grouped them based on a high-level view of the experiment requirements. Some of these classifications, or experiment patterns, seemed common enough to be used as drivers for the PRECIP development. In this paper we present a subset of the patterns categorized as follows: (i) domain focused, (ii) networking, (iii) mapReduce, and (iv) educational.

Bioinformatics projects, for example, aim at developing and improving methods for storing, retrieving, and analyzing biological data. A major activity in bioinformatics projects is to develop software tools to generate useful biological knowledge. These projects have a wide range of applications including image and signal processing for extraction of results from large amounts of raw data; sequencing and annotating genomes and their observed mutations; textual mining of biological literature and the development of biological and gene ontologies to organize and query biological data; and simulation and modeling of DNA, RNA, and protein structures. In the context of these projects, PRECIP can be used, for instance, for testing software tools and techniques, sending and receiving files to/from the cloud, or for running biological extractions of knowledge and data.

Networking projects are, in general, primarily about simulation and improving networks, peer-to-peer networks, sensor networks, and social networks. In such projects there may be some methods for testing enhancements either manually or automatically. PRECIP can be used for simulating networking scenarios and for running dynamic network experiments on the cloud, such as removing and/or adding nodes during an experiment execution to evaluate routing strategies. PRECIP can also be used to perform algorithm evaluations and comparisons across infrastructures.

MapReduce is a programming model for processing large datasets with a parallel, distributed algorithm on a cluster [8]. Projects using MapReduce are basically about analysis, enhancement and performance evaluations of systems in MapReduce environments, fault management, predictive systems, and use of Hadoop [9] as one of the implementations of MapReduce. The purpose of these projects is to present an improved version of MapReduce in different scientific areas. A subset of FutureGrid projects related to MapReduce are "Climate Data Analytics Using MapReduce", in which PRECIP can be used as an API to execute climate data experiments on a workflow management system to compare the execution time and I/O file sizes against their proposed methods. Besides, file system benchmarking can also be done. For instance, in the "Wide area distributed file system for MapReduce applications on FutureGrid platform" FutureGrid project, PRECIP may be used for benchmarking the proposed file system for comparison with the shared file system.

Educational projects include course projects, understanding of the uses for FutureGrid and similar cloud computing services, data analysis, visualization and simulation of real world experiments in distributed computing, real time prediction, comparison of different cloud infrastructures, and systems evaluations. In this context, PRECIP can be used as a tool for ease of access to cloud infrastructures and testing scenarios used in this area. It can also be used for benchmarking different infrastructures and performance comparisons. Projects such as "Cloud Computing In Education" and "Experiments in Distributed Computing" may use PRECIP to teach the interaction between users and the cloud, and to see how virtual machine (VM) instances can be assembled for further collaborations.

III. PRECIP Design

Figure 1 shows an overview of the PRECIP's architecture. We use a layered architecture presented as concentric regions, where the user interface is represented in the center layer, the internal experiment layer represents the steps involved in an experiment, the external layer represents the components used to manage resources, and the surrounding layer contains grouped sets of cloud instances. Shaded areas around instances denote tag groups.

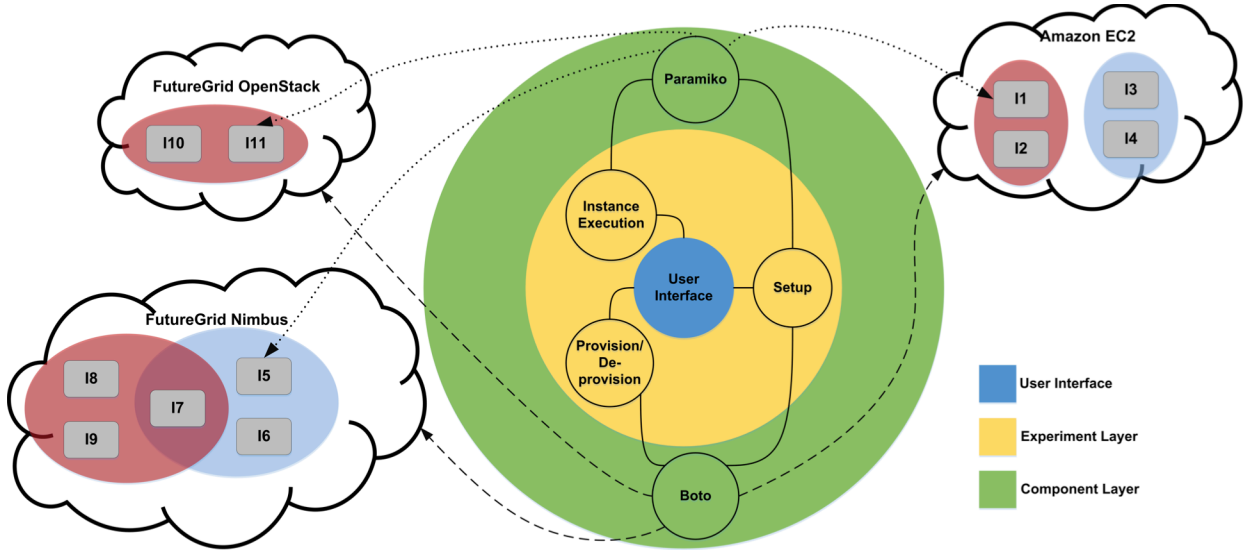

Figure 1: Overview of PRECIP's Architecture. The figure shows the functional layers in the system, starting with the user in the center, instance management in the next layer, and supporting libraries (Boto and Paramiko) as the outer layer. The figure also highlights the fact that PRECIP can concurrently interact with multiple cloud infrastructures, and how instance tags can be used to create logical subsets of running instances.

Running PRECIP experiments in a distributed environment is a simple 4-step procedure: (i) experiment setup, (ii) resource provisioning, (iii) experiment execution, and (iv) resource de-provisioning. We detail these steps, and some basic PRECIP features, in the following subsections.

A. *Virtual Machine Images*

In PRECIP, researchers can use their own virtual machine images (VMI). These images do not need to contain any PRECIP-specific software; otherwise porting an experiment to PRECIP would not be affordable. Any interaction with instances is done over SSH connections and any standard Unix-like VMI can be used. An experiment can also use cloud-provided base VMIs, and for more complex experiments, more complex VMIs can be used. This feature enables researchers to use VMIs that require special software stacks or custom kernels. An alternative to a complex VMI is to use the PRECIP API to run bootstrap scripts on the VMIs to install or configure the required software.

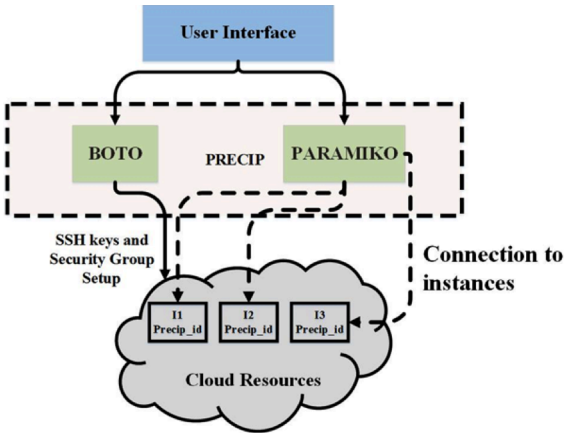

Figure 2: PRECIP uses the Boto library to communicate with the cloud infrastructures and the Paramiko library to run commands on running instances.

B. *Instance Tags*

Another important feature of PRECIP are tags. Instance tagging is a powerful and flexible concept, which simplifies interactions with a potentially large number of instances. The concept of Tags was first used in an interactive experiment system on FutureGrid [2]. Tags are used throughout PRECIP to identify, manipulate and interact with instances. Arbitrary tags can be associated with an instance during its provisioning phase; these tags are global and available during the whole experiment. A tag describes a logical group of instances, so that API methods such as running a remote command or copying files can be performed to all instances matching a given tag. Tag groups are illustrated in Figure 1 by shaded ovals. Note that an instance can have an arbitrary number of tags, and the researcher can use the tags to create subsets, which can intersect in any number of ways.

C. *Experiment Setup*

Internally to PRECIP, setting up an experiment consists of three phases: (i) establishing a new connection to the cloud endpoint, (ii) registering a pair of SSH keys for internal connections, and (iii) setting up a default security group. All that a researcher needs to provide is an endpoint, a region, and their security credentials; then PRECIP sets up the experiment on behalf of the researcher.

PRECIP manages communication with cloud instances in a separate context from the user's local system. This ensures that the experiment management does not interfere with an existing cloud setup of the system. Communication is performed through SSH key pairs and security groups. The latter can be considered as the firewalls of cloud instances. PRECIP uses the Boto [10] and Paramiko [11] libraries to communicate with the cloud providers and to manage SSH connections to the VM instances.

Boto provides an integrated API to access cloud services. It includes support for the Amazon EC2 and private cloud systems such as Eucalyptus, OpenStack, and Open Nebula that implement EC2 compatible interfaces. Paramiko provides

secure (encrypted and authenticated) connections to remote instances through the SSH2 protocol.

Figure 2 shows how Boto and Paramiko are used to set up an experiment in PRECIP. On PRECIP's first execution, a PRECIP specific SSH key pair is created and stored on researcher's local machine. On the remote cloud infrastructure, the key pair is automatically registered and a PRECIP security group is created. A security group is defined as a named collection of access rules. For instance, if an experiment requires a specific set of opened ports, the rules that infer those ports can be added to the PRECIP security group. By default, the PRECIP security group is fully open to accept all connections the experiment may require.

D. Resource provisioning/ de-provisioning

The second step to perform an experiment is resource provisioning. In this step, a large number of instances can be started at the same time. Internally, PRECIP uses Boto for resource provisioning. To provision an instance, all a researcher has to do is to provide a VMI identification, the number of instances and their types, and an arbitrary number of tags used to identify the instance later.

A common problem, especially for academic clouds, is that cloud instances may not start correctly. They may fail to boot correctly, may reach a timeout during boot, or may not be able to properly execute instance setup scripts, which could indicate that the networking was not set up correctly for the instance. As a result, in the provisioning phase, whenever an instance has initialization problems, PRECIP will automatically retry the provisioning operation up to the specified maximum number of retries. This fault-tolerant mechanism enables an experiment to autonomously recover from initialization failures, and prevents the entire experiment from being aborted due to provisioning faults. During the experiment, the researcher can de-provision instances by using tags, a feature commonly used in experiments when testing fault tolerance and fail over of third party software. At the end of the experiment, all instances are de-provisioned automatically.

E. Experiment Execution

Once all instances are initialized and configured, PRECIP uses Paramiko to give the researcher the capability to effortlessly run commands on the subset of provisioned virtual machines, and copy files to/from subsets of instances identified by tags. Figure 3 shows an example where two files are uploaded to instances. The first operation requests a file transfer to instances tagged as `Tag A`, while the second to instances tagged as `Tag C`.

Figure 4 shows an example of an experiment performed on multiple infrastructures (Amazon EC2 and FutureGrid OpenStack in this case). The experiment starts by setting up a SSH connection between the user and the virtual machine instances (`step 1`). Then, two virtual machines are provisioned: one tagged as `[A,T]` to Amazon EC2, and another tagged as `[B,T]` to OpenStack. As initialization times may differ among infrastructures, PRECIP waits until

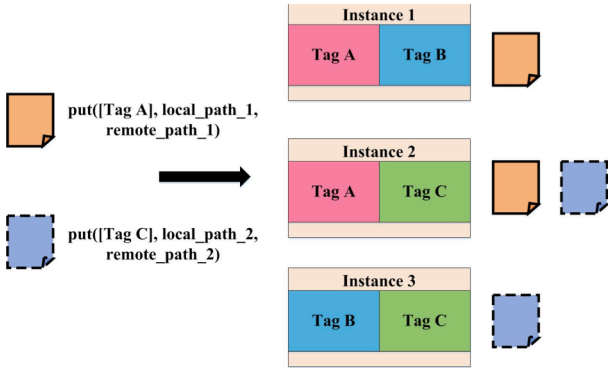

Figure 3: Uploading files to subset of instances identified by instance tags.

all instances are initialized (i.e. all instances are in `Ready` status) to perform computations (`step 3`).

The first operation pushes two files into the provisioned instance. Note that each file is transferred according to the specified tag on the request (`step 4`). In `step 5`, a run command is used to execute a shell script on instances tagged as `[T]`. Now, both instances execute the same operation producing result files that are transferred to the researcher's machine (`step 6`). At the end of the experiment, instances matching the tag `[T]` are de-provisioned.

F. Logging

To provide a complete record of the experiment, all major events in PRECIP get logged to the console using Python's standard logging framework. The events are time stamped in the ISO 8601 format (`YYYY-MM-DD hh:mm:ss`). Instance provisioning events include time when instances are requested, time out, fail or are fully provisioned and ready to use, and when an instance de-provisioning is requested. File transfers and remote command executions are logged as well. Researchers can add their own events simply by also using the Python logging framework. There is a discussion in the PRECIP community to possibly integrate Netlogger [12] to support more extensive and structured experiment logging.

G. PRECIP API

As mentioned earlier little Python knowledge is required for using PRECIP. In this subsection, we provide an overview of the most common parts of the API. To start the cycle of providing new instances, researchers have to provide the image ID, instance type, number of instances, tags, and optionally timeout and max retires:

```
provision(image_id, instance_type, count,
          tags, boot_timeout,
          boot_max_tries)
```

The `provision` function does not block for the instance to finish booting. Therefore, we use a separate `wait()` method that acts as a barrier for all instances, blocking them until all instances have finished booting and are accessible via their external hostnames:

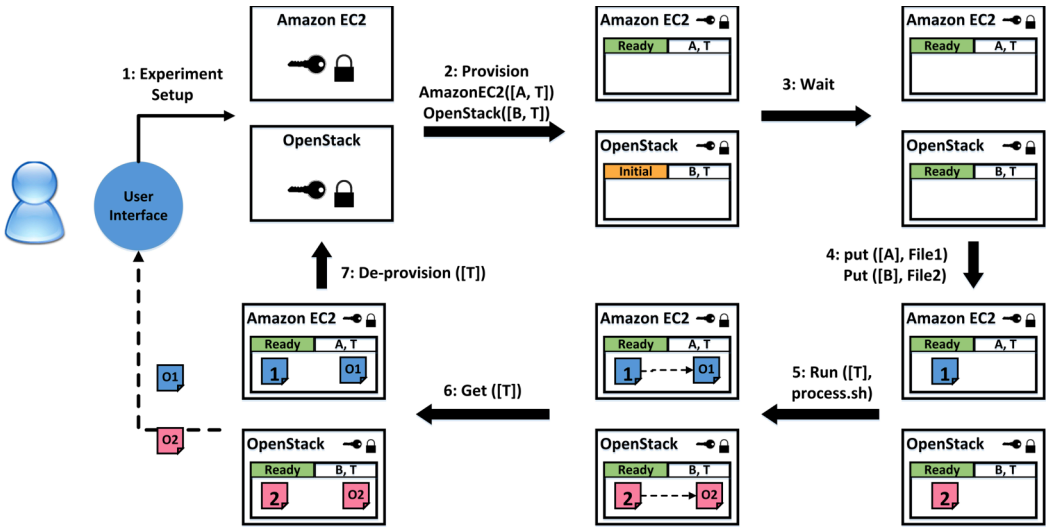

Figure 4: Sample experiment across multiple cloud infrastructures.

```
wait(tags=[])
```

The API contains methods for transferring files from or to a set of remote machines matching the tags:

```
get(tags, remote_path, local_path, user)
put(tags, local_path, remote_path, user)
```

For running commands on the instances matching the tags we use:

```
run(tags, cmd, user)
```

For copying a script from the local machine to the remote instances and executing the script:

```
copy_and_run(tags, local_script, user)
```

The `deprovision(tags)` function terminates all the provided instances and clean up after itself.

IV. Evaluation

In this section we use a sample experiment to evaluate PRECIP. In the first subsection, we present error categories we faced when running experiments. In the second subsection, we explain the time durations consumed by each part of the experiment, and we assess overhead times added by PRECIP to the overall experiment execution time. We also show details of how PRECIP detects infrastructure faults on OpenStack, how it addresses such issues, and finally recovers from them. The PRECIP script used for this experiment is available for download and reproducibility at http://pegasus.isi.edu/precip/paper2013/.

A. Error Classification

There are two types of errors that commonly occur when performing an experiment: (i) infrastructure and (ii) communication related errors. Infrastructure errors include exceeding the maximum number of allowed instance requests for a cloud infrastructure, and internal server failures, which lead to timeouts. Communication errors are related to the inability to establish a connection to an instance, and getting errors due to overloaded services.

B. Overhead Characterization

The execution of a PRECIP experiment is steered by a provisioning phase, the experiment execution itself, and a de-provisioning phase (DP). As it is shown in Figure 5, the time consumed by the provisioning phase is an aggregation of five time intervals. The waiting time for the instance response (WTIR) includes the time span from the instant PRECIP connects to the endpoint, and the instant the instance is assigned to the user; the waiting time added by the `wait()` method (`step 3` in Figure 4); and, the time to establish a SSH connection to the instance. The waiting operation adds an overhead to the experiment execution that may be negligible if instances initialize successfully, but may be important if some instances have failures during their initialization phase. As described in the previous section, several errors may occur during an experiment execution, and then fault-tolerant techniques are applied to mitigate these failures. However, these mechanisms may add overheads to the experiment execution. We characterize these overheads as the time to discover errors (TDE), and the time to retry an operation (TRO). If the experiment requires complex images, bootstrap scripts can be run by the experiment API on the images to install or configure required software. This time partition is identified as the time to run bootstrap scripts on the VM instances (TRBS). Finally, the provisioning phase ends when instance initializations are accomplished so that instances are ready and bootstrapped (TRB).

Table 1 shows the metrics that we have evaluated for an

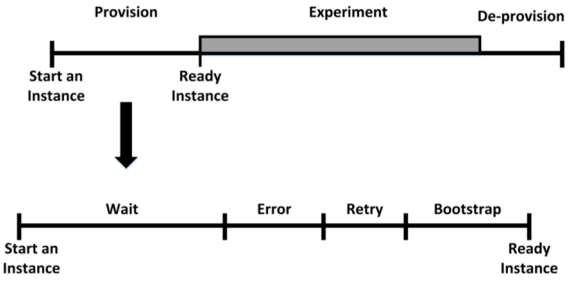

Figure 5: Time Portions of an Experiment.

experiment in which 20 VM instances are set up on Amazon EC2. After all the instances have booted, we run a simple "hello world" experiment and then de-provision the instances.

Table 1: Metrics Evaluation on Amazon EC2 (Times are in milliseconds)

Metrics	Average	Std. Dev.	Min	Max.
WTIR	3879	802	2214	5222
TRBS	538	287	66	945
TRB	473	320	22	989
TDE	-	-	-	-
Number of Failures	0	-	-	-
Number of Retries	0	-	-	-
De-provision Time	451	290	12	945

Table 2 shows the same metrics as in Table 1 on OpenStack (FutureGrid's India resource). Comparing both executions, Amazon EC2 performs better in metrics such as De-provision Time, while India performs better in metrics such as WTIR and TRB.

Table 2: Metrics Evaluation on OpenStack on India (Times are in milliseconds)

Metrics	Average	Std. Dev.	Min	Max.
WTIR	1112	445	225	1809
TRBS	479	306	33	989
TRB	405	255	8	901
TDE	-	-	-	-
Number of Failures	0	-	-	-
Number of Retries	0	-	-	-
De-provision Time	538	293	66	993

Running experiments using PRECIP on OpenStack cleared some infrastructures issues that were mainly due to Boto having a problem with Nova when assigning a security group to booting instances. Thus, running experiments on India led to failures in the network setup for several instances and all those instances got timeout with error code 500, which is internal server error.

After investigating the cause of those failures, we figured out that the bug is infrastructure related. Since India's OpenStack is Essex version, and the volume service is Nova, it turned out that Nova has problem with assigning security groups to the starting instances. For the experiment, we temporarily suppressed the assigning of security groups to the instances on OpenStack until India's upgrade. After starting instances without security groups, all instances booted successfully and we were able to run our experiment on those instances.

V. SAMPLE EXPERIMENT

In this section we illustrate the use of PRECIP for database scalability experiment. The goal of this experiment is to evaluate the performance of various distributed configurations of MongoDB [13]. Document-oriented databases can be used as alternative to traditional relational databases or distributed file systems for storing and processing large amounts of data by scientific applications. Examples come from bioinformatics, where MongoDB can be used as a convenient way of storing results of genetic sequence similarity searches [14] or for data analysis in materials science [15]. In this experiment we evaluate the scalability of MongoDB in the

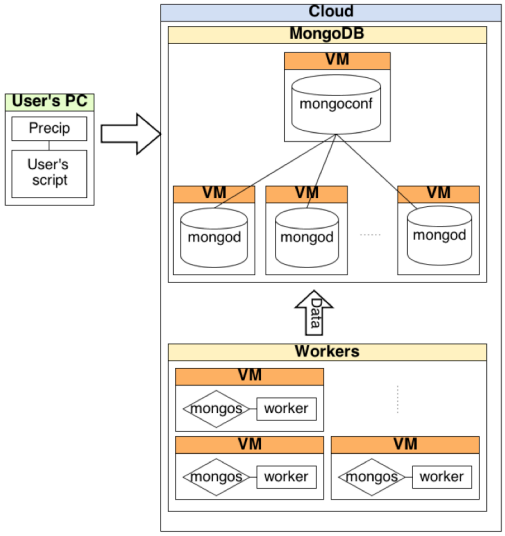

Figure 6: Conceptual diagram showing components used in MongoDB experiment.

configuration where the data is partitioned between multiple MongoDB servers, called shards, to handle the increasing load of concurrent clients (e.g. compute jobs on a cluster).

As shown in Figure 6, the experiment consists of two logical machine groups. The 'MongoDB cluster' group includes VMs hosting the DB. The 'Workers' group is responsible for processing and pushing data to database. All the VMs are hosted in the same cloud and are managed by a single PRECIP script.

Hosts from the first group run MongoDB applications to create one distributed database. They always include one low-spec virtual machine for running the Mongo configuration service and a variable number of hosts responsible for shards, running the 'mongod' program connected to the configuration service.

The second group consists of machines responsible for reading, parsing and storing bioinformatics experiment results in MongoDB. Each machine runs the 'mongos' program, which acts as a routing service for the sharded database. The custom 'worker' native application takes a list of documents as an input and reads them in order to extract the data and store it in the database using mongos.

The 'User script' component is a single program written in Python that uses PRECIP library to connect to cloud providers, create instances, and set up them. The script is available for download and reproducibility at http://pegasus.isi.edu/precip/paper2013.

The experiment implemented in PRECIP has the following steps.

1. In the **boot phase** the machines are created and the script waits until all machines can be accessed via SSH protocol.
2. During the **setup phase** all the programs are installed, compiled if necessary, and configured. This bootstrap phase assumes that all the VMs are running

basic Centos Linux. The script installs MongoDB using RPM packages, pushes the sources of the worker application to be compiled in the target environment, and downloads input files. At the end of this phase the monitoring tools, including `sysstat` and `mongostat`, are started.

3. After the setup is done the experiment enters the **test phase** where worker applications start to process and push the data to the DB.
4. When the workers complete, the **upload phase** is initiated and data collected during the experiment is pushed to the Amazon S3 storage service. This data consists of monitoring data and worker application results.
5. At the end of the experiment, the **cleanup phase** is started to ensure that all resources are freed and that experiment execution measurements are saved.

We ran this experiment on Amazon EC2 and on a FutureGrid OpenStack cluster. Example results are shown in Figure 7 and Figure 8. Figure 7 shows the execution times of a sample experiment, broken into phases. Figure 8 shows example results of measured insert ratio for the configuration using 2 shards. In this particular setup, we observe that the write speed was not constant during the test and the shards were not equally loaded. This means that MongoDB was not able to take full advantage of the sharded configuration. Using PRECIP allows us to repeat the experiments multiple times to identify such performance problems and fine tune the configuration parameters.

Conducting the experiment in cloud environment has several advantages. One reason behind it is that it requires complex setup consisting of many different services depending on each other. In addition, most of the setup involves operations applied to the host OS (starting services, editing configuration files, etc.). This cannot be done easily in a classical cluster or grid environment, where security settings are strict and the environment is not configurable. However, the drawback of the cloud environment is the need for another layer managing virtual appliances (starting, installing, stopping virtual machines etc.).

Using PRECIP, which provides such a layer, results in faster and simpler experiment development by simplifying the process of acquiring, configuring, and releasing the resources. Using PRECIP also enables users to easily switch between cloud providers. The experiment can be reproduced using many different environments located within independent clouds, providing information not only about MongoDB performance, but also about cloud performance.

VI. Related Work

Cloud management tools from the experiment point of view are categorized in two ways: (i) provisioning tools and (ii) experiment management tools.

Provisioning tools are those that are developed for automatic provisioning, configuration and management of virtual machine instances, these tools provide allocated individual VMs with configuration of CPU, memory, disk space and so on. On the other hand, experiment management

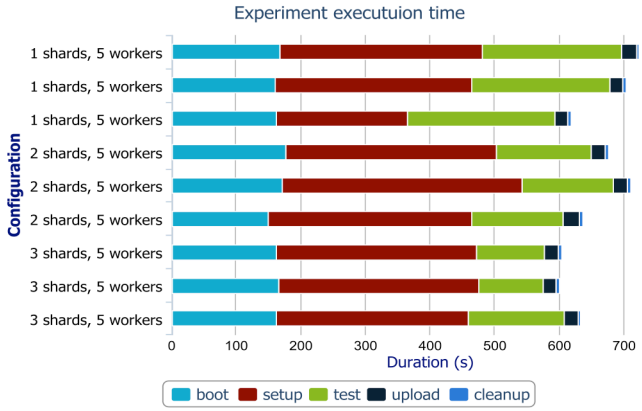

Figure 7: Example execution times of all phases of MongoDB experiment on Amazon EC2.

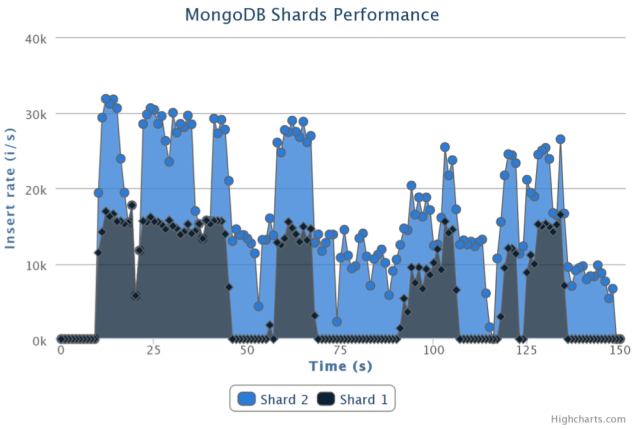

Figure 8: Example measurements of insert ratio of MongoDB, in configuration with 2 shards and 5 workers.

tools are those, which include automatic provisioning tools, experiment running, and de-provisioning of assigned instances. The goal of provisioning tools is to develop a capability for automatically deploying virtual machine images in the cloud. The overall systems should allow for the dynamic configuration of the images, so that a variety of services can be deployed based on the needs of the user. An example system of this type is Wrangler [16] that automatically provisions and configures virtual clusters in the cloud. It is used to provision virtual clusters for workflow applications on Amazon EC2, the Magellan cloud at NERSC, the Sierra and India clouds on the FutureGrid, and the Skynet cloud at ISI. Wrangler requires the desired cloud setup to be statically defined in an XML file and does not allow the experiment to easily modify the configuration, for example adding or removing instances. It also requires some services to be installed on the VM image, which can be a burden for users. Another potential drawback with Wrangler is that it is not fault tolerant and assumes once a node is started it will not fail, but a single failure on any VM instances will lead to unusable clusters and at the same time being charged by commercial cloud service providers.

Experiment management tools add more overheads to the systems but at the same time allow users to automatically run

different kinds of scientific experiments. One example of these tools is Zoo [17], a desktop experiment management environment. Zoo mainly focuses on data-intensive experiments and is interested mostly in data collection, exploration and database management.

There are a few other projects, which provide some of the features that PRECIP does. Moses [18], the Experiment Management System, is used for preparing training data and building, tuning, testing, scoring, and analyzing language and translation models. B-Fabric [19] is an all-in-one tool for the integrated management of experimental and annotation data in the life sciences. B-Fabric is a data management system, which is mainly used for data intensive experiments while our system covers both data intensive and computationally intensive experiments. Each project concentrates on different user communities, so comparisons with our approach are not always relevant. In general, the most important aspects of our idea are its generic nature to run experiments across multiple cloud infrastructures, having control over instances using tagging, automatic handling of provisioning, being fault tolerant via retrying to boot new instances in case of failures, setup of SSH keys and security groups, and ability to repeatedly run the same experiment and gather results. Many of the other projects are not as focused on generic experiment management as PRECIP is. For example, B-Fabric is a complete project management tool, of which experiment management is just one part, and some of its concepts are life science specific.

VII. Conclusions

In this paper we have introduced PRECIP features and concepts, and how it provides a generic cloud experiment management tool for computer science based experiments, cloud systems evaluations, and scalability experiments. The goal of designing this API is to prepare the foundation for repeatable, shareable and peer reviewable experiments. We presented experiment domains PRECIP can be used for, the details of experiment execution, and provided and overview of the important methods of the API. We then demonstrated PRECIP features by performing a simple instance provisioning and database scalability experiment.

Looking forward, we are exploring adding several new features to PRECIP, such as automatic gathering and comparing performance information of experiments, and providing a set of modules for common tasks such as HTCondor, Hadoop, and shared file system setups. We would also like to be able to handle stale instances, which are not terminated correctly in previous experiments. We are planning to help running and testing experiments in FutureGrid projects as well as improve logging feature of the system by using other logging frameworks such as NetLogger.

VIII. Acknowledgment

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed." Partners in the FutureGrid project include U. Chicago, U. Florida, San Diego Supercomputer Center - UC San Diego, U. Southern California, U. Texas at Austin, U. Tennessee at Knoxville, U. of Virginia, Purdue U., and T-U. Dresden. We would also like to express our thanks to Tomasz Gubała for his help with MongoDB experiments. The use of Amazon EC2 resources was supported by an AWS in Education research grant.

IX. References

[1] Y. Jiang, C. Perng, T. Li, and R. Chang, "ASAP: A Self-Adaptive Prediction System for Instant Cloud Resource Demand Provisioning," *IEEE 11th International Conference on* in *Data Mining (ICDM),* 2011, pp. 1104–1109.

[2] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo, S. Smallen, W. Smith, and A. Grimshaw, "FutureGrid - a reconfigurable testbed for Cloud, HPC, and Grid Computing," in *Contemporary High Performance Computing: From Petascale toward Exascale*, J. Vetter, Ed. Chapman & Hall, 2013.

[3] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, G. G. Pike, W. Smith, J. Vöckler, R. J. Figueiredo, J. Fortes, and K. Keahey, "Design of the FutureGrid experiment management framework," in *Gateway Computing Environments Workshop (GCE), 2010*, 2010, pp. 1–10.

[4] "FutureGrid Project Statistics," Jul-2013. [Online]. Available: https://portal.futuregrid.org/projects-statistics.

[5] "OpenStack," Jul-2013. [Online]. Available: http://www.openstack.org/.

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Cluster Computing and the Grid, 2009. CCGRID'09*, pp. 124–131.

[7] K. Keahey and T. Freeman, "Nimbus or an Open Source Cloud Platform or the Best Open Source EC2 No Money Can Buy.," Supercomputing 2008.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun Acm*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[9] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[10] "*Boto: A Python interface to Amazon Web Services*," Sep-2013. [Online]. Available: https://github.com/boto/boto.

[11] "Paramiko: Python SSH module," Jul-2013. [Online]. Available: https://github.com/paramiko/paramiko.

[12] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter, "The NetLogger Methodology for High Performance Distributed Systems Performance Analysis," in *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, USA, 1998, p. 260–.

[13] "MongoDB," Jul-2013. [Online]. Available: http://www.mongodb.org/.

[14] M. Malawski, J. Meizner, M. Bubak, and P. Gepner, "Component Approach to Computational Applications on Clouds," *Procedia Comput. Sci.*, vol. 4, pp. 432 – 441, 2011.

[15] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a MongoDB and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, New York, NY, 2013, pp. 13–20.

[16] G. Juve and E. Deelman, "Wrangler: virtual cluster provisioning for the cloud," in *Proceedings of the 20th international symposium on High performance distributed computing*, NY, USA, 2011, pp. 277–278.

[17] Y. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti, "ZOO: A Desktop Experiment Management Environment," in *In Proc. 22nd International VLDB Conference*, 1996, pp. 274–285.

[18] "Moses - Experiment Management System," Jul-2013. [Online]. Available: http://www.statmt.org/moses/?n=FactoredTraining.EMS.

[19] C. Türker, E. Stolte, D. Joho, and R. Schlapbach, "B-Fabric: A Data and Application Integration Framework for Life Sciences Research," in *Data Integration in the Life Sciences*, vol. 4544, S. Cohen-Boulakia and V. Tannen, Eds. Springer Berlin Heidelberg, 2007, pp. 37–47.