Comparison of Multiple Cloud Frameworks

Gregor von Laszewski, Javier Diaz, Fugang Wang, Geoffrey C. Fox
Pervasive Technology Institute, Indiana University
Bloomington, IN 47408, U.S.A.
e-mail: {laszewski, javier.diazmontes, kevinwangfg, gcfexchange}@gmail.com

***Abstract*—Today, many cloud Infrastructure as a Service (IaaS) frameworks exist. Users, developers, and administrators have to make a decision about which environment is best suited for them. Unfortunately, the comparison of such frameworks is difficult because either users do not have access to all of them or they are comparing the performance of such systems on different resources, which make it difficult to obtain objective comparisons. Hence, the community benefits from the availability of a testbed on which comparisons between the IaaS frameworks can be conducted. FutureGrid aims to offer a number of IaaS including Nimbus, Eucalyptus, OpenStack, and OpenNebula. One of the important features that FutureGrid provides is not only the ability to compare between IaaS frameworks, but also to compare them in regards to bare-metal and traditional high-performance computing services. In this paper, we outline some of our initial findings by providing such a testbed. As one of our conclusions, we also present our work on making access to the various infrastructures on FutureGrid easier.**

Cloud, Grid; Nimbus; Eucalyptus; OpenStack; OpenNebula; RAIN; FutureGrid

I. INTRODUCTION

Cloud computing has become an important tool to deliver infrastructure as a service (IaaS) to users that require a great deal of customization and management of their own software stacks. In addition, we observe that users demand further abstraction and expect platforms as a service (PaaS) to be readily available to conduct higher-level development efforts. Together IaaS and PaaS can provide potent solutions not only to business users, but also to the educational and scientific computing communities. We observe that in the scientific community we can distinguish a number of typical user categories based on their usage in regards to scale, virtual machines used, computational tasks, and the number of users served by the services. In particular we are interested in the following use cases.

First, we distinguish users that demand very few images but want to run them for a long period of time with guarantees on high-availability. Such environments are targeted to support what is today termed the "long tail of science". In this case, many millions of scientific users with moderate computing needs benefit from the delivery of less compute intense services.

Second, we distinguish scientists requiring a large number of resources to conduct actual calculations and analyses of data to be exposed to their community. This is the traditional high-performance computing use case.

Third, we identified a class of users with requirements in-between the two previous cases. These users have modest but still significant demand of compute resources. Hence our use cases motivate three classes of infrastructures:

1. Traditional high performance computing (HPC) services offered as part of traditional HPC centers.
2. Hosting services for production such as services that cater to a community, including gateways, Web servers and others. Such services may interface with services that run on traditional HPC resources.
3. Access to moderate compute resources to support many moderate calculations to its many users that demand it.

Within this paper we focus on the third class of users. We will identify the rational for some of the choices that we offer in FutureGrid and identify how to simplify access to an environment that provides so many choices.

The paper is structured as follows. In Section II, we present an overview of FutureGrid and explain why FutureGrid provides the current services it offers. We also outline some future directions motivated by simple usage patterns observed in FutureGrid. One of the main objectives of this paper is to contrast the different IaaS frameworks we offer and project first results of our qualitative comparison described in Sections III and IV, respectively. Based on our qualitative analysis we strive to provide answers to the following questions:

1. *Which IaaS framework is most suited for me?*
2. *How can I compare these frameworks not just between each other, but also to bare-metal?*

The later is of special interest as at this time many of the Cloud frameworks are still under heavy development and pathways to utilize multiple of them are of current interest.

In Section V, we impart our thoughts on providing PaaS offerings attractive for our user communities. Next, in Section VI, we discuss what implications this multitude of service offerings has for the user community. In Section VII we introduce rain and in Section VIII we detail the scalability tests performed in different FG infrastructures. Finally, Section IX collects the conclusions of this work.

II. FUTUREGRID

The FutureGrid project is sponsored by NSF and includes partners from Indiana University, University of Chicago, University of Florida, San Diego Supercomputing Center, Texas Advanced Computing Center, University of Virginia, University of Tennessee, University of Southern California, Dresden, Purdue University, and Grid 5000.

Among its sites Futuregrid has a set of distributed resources totaling about 5000 compute cores. Resources include a variety of different platforms allowing users to access heterogeneous distributed computing, network, and storage resources. This variety of resources and services allow interesting interoperability and scalability experiments. Additionally, FG provides a fertile environment to explore a variety of IaaS and PaaS frameworks.

Early on we were faced with two elementary questions: (1) Does a user need access to more than one IaaS framework, and (2) if so, which frameworks should we offer? To answer these questions, we analyzed data gathered from the user community of FutureGrid as part of the project registration process. Each project requestor had to provide us feedback on the technologies that were relevant for the execution of their projects. The result of this information is depicted in Figure 1. Figure 2 shows the demographic of the projects we host on FutureGrid classified by scientific disciplines and goals. We observe the following:

1. Nimbus and Eucalyptus were requested the most. This is not surprising, as we made most advertisement for these systems and recommended them for educational class projects on FG. Originally we had more requests for Eucalyptus but scalability issues of the Eucalyptus install on FutureGrid resulted in a recent increase in Nimbus requests.
2. High Performance Computing (HPC) services are requested as third highest category. This is motivated by our affiliation with traditional HPC communities as well as the strong ties to XSEDE.
3. Hadoop and map/reduce were requested by about 36.78% of the projects. This number is higher than the one reported in Figure 1, as we combined the values for Hadoop and MapReduce while only considering unique entries.
4. We saw recently an increase in requests for OpenStack. It has just become one of the preferred open source solutions for cloud computing within a large number of companies, but also within the research community.
5. We have seen an increased demand for the support of OpenNebula. It is quite popular as part of the European Cloud efforts and has gained substantial backing also by US projects such as Clemson University and Fermi Laboratory as part of their cloud strategies.
6. Not surprisingly the largest contingent of our users is the group of technology experts.

Based on this analysis we spent our effort to enable such services within FutureGrid. As a result we are currently providing the following partitioning between services as listed in Figure 3. However, the number of nodes associated between these services can be changed by request.

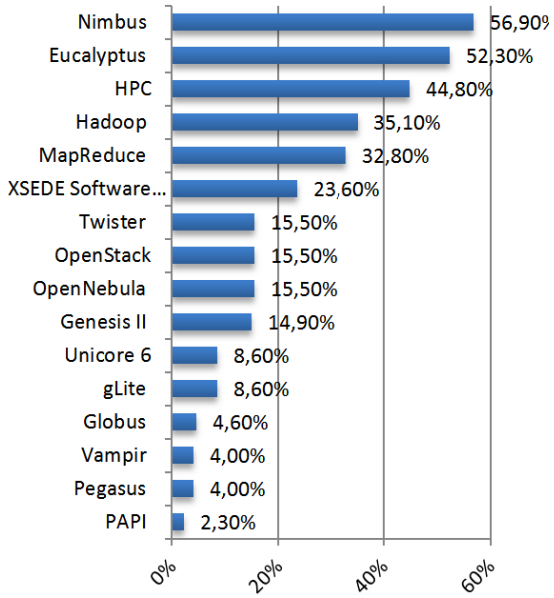

Figure 1: Technology choices made as part of the project application process in FutureGrid. Note that multiple entries could be selected so the total will be more than 100%.

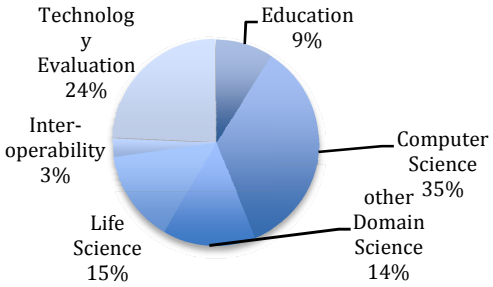

Figure 2: The distribution of the scientific areas that we identified while reviewing the project requests. (Note that 20 project have yet to be integrated into this data).

OpenNebula does not appear on this chart because we have not made it officially accessible. Nevertheless, we have conducted scalability experiments (see Section IV) and monitored usage patterns that could motivate a possible shift in our current deployment strategy. For this decision we are taking into account also other factors, such as what our users do on the portal. An interesting observation we made is that our OpenNebula tutorial that we host on the FG portal is one of our most popular tutorials (see Figure 3) although we do not offer OpenNebula on FutureGrid at this time. This prompts us to identify if we should also offer OpenNebula.

At present, we are integrating mechanisms to float resources between the IaaS frameworks. Thus, today we could decide to assign some resources to Nimbus, while tomorrow the same resources could be assigned to another IaaS farmework Hence, the different infrastructures of our testbed can be resized on-demand to support scalability experiments or scientific problems with high resource requirements.

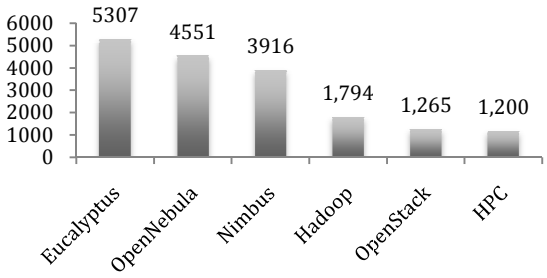

Figure 3: Number of hits on our tutorial pages

Resource	HPC	Eucalyptus	Nimbus	Openstack	
IU-INDIA (920 cores)	**54.8%** (504 cores)	**20.9%** (192 cores)			HPC(54.8%) Mgm(0.9%) Eucalyptus(20.9%) xperimental(23.5%)
IU-XRAY (672 cores)	**100%** (672 cores)				HPC(100%)
TACC-ALAMO (760 cores)	**84.2%** (640 cores)		**15.8%** (120 cores)		Nimbus(15.8%) HPC(84.2%)
UC-HOTEL (624 cores)	**48.7%** (304 cores)		**51.3%** (320 cores)		Nimbus(51.3%) HPC(48.7%)
UCSD-SIERRA (624 cores)	**44.9%** (280 cores)	**19.2%** (120 cores)	**23.1%** (144 cores)	**3.8%** (24 cores)	Mgm(6.4%) Nimbus(23.1%) Openstack(3.8%) HPC(44.9%) Eucalyptus(19.2%) Experimental(2.6%)
UFL-FOXTROT (216 cores)			**96.3%** (208 cores)		Nimbus(96.3%) Mgm(3.7%)

Figure 4: Typical partitioning of FG compute resources by IaaS framework

III. Overview of Cloud IaaS Frameworks

One fundamental concept in cloud computing is based on providing Infrastructure as a Service (IaaS) to deliver resources to customers and users instead of purchasing and maintaining compute, storage, and network. Typically, this is achieved by offering virtual machines to the users. In order to establish such a service, a number of frameworks are available including Eucalyptus, Nimbus, OpenNebula, OpenStack (see Figure 4). Next, we provide a short discussion about these frameworks and outline some major qualitative differences between them.

A. Nimbus

The Nimbus project [1] is working on two products that they term *"Nimbus Infrastructure"* and *"Nimbus Platform"*.

Nimbus Infrastructure: The Nimbus project defines the Nimbus Infrastructure to be "an open source EC2/S3-compatible Infrastructure-as-a-Service implementation specifically targeting features of interest to the scientific community such as support for proxy credentials, batch schedulers, best-effort allocations and others." To support this mission, Nimbus is providing its own implementation of a storage cloud compatible with S3 compatible and enhanced by quota management, EC2 compatible cloud services, and a convenient cloud client which uses internally WSRF.

Nimbus Platform: The Nimbus platform is targeting to provide additional tools to simplify the management of the infrastructure services and to facilitate the integration with other existing clouds (OpenStack and Amazon). Currently, this includes the following tools a) cloudinit.d coordinates launching, controlling, and monitoring cloud applications, b) a context broker service that coordinates large virtual cluster launches automatically and repeatedly [1, 2]

B. OpenNebula

OpenNebula [3, 4] is an open source IaaS toolkit. Its design is flexible and modular to allow integration with different storage and network infrastructure configurations, and hypervisor technologies [5]. It can deal with changing resource needs, resource additions, live migration, snapshotting, and failure of physical resources [3, 6]. Furthermore, OpenNebula supports cloud federation to interface with external clouds, which provides scalability, isolation, and multiple-site support. This lets organizations supplement the local infrastructure with computing capacity from public clouds to meet peak demands, or implement high availability strategies. Thus, a single access point and centralized management system can be used to control multiple deployments of OpenNebula.

OpenNebula supports different access interfaces including REST-based interfaces, OGF OCCI service interfaces, and the emerging cloud API standard, as well as the de-facto AWS EC2 API standard.

The authorization is based on passwords, ssh rsa keypairs, X.509 certificates or LDAP. This framework also integrates fine-grained role –based ACLs that as part of its authorization capabilities.

The authorization is based on passwords, ssh rsa keypairs, X.509 certificates or LDAP. This framework also supports fine-grained ACLs to manage users with multiple roles as part of its authorization capabilities.

The storage subsystem supports any backend configuration, from non-shared file systems with image transferring via SSH to shared file systems (NFS, GlusterFS, Lustre) or LVM with CoW (copy-on-write), and any storage server, from using commodity hardware to enterprise-grade solutions.

C. OpenStack

OpenStack [7] is a collection of open source components to deliver public and private clouds. These components currently include OpenStack Compute (called *Nova*), OpenStack Object Storage (called *Swift*), and OpenStack Image Service (called *Glance*). OpenStack is a new effort and has received considerable momentum due to its openness and the support of companies.

Nova is designed to provision and manage large networks of virtual machines, creating a redundant and scalable cloud computing platform. Swift is used to create a redundant, scalable object storage using clusters of standardized servers in order to store petabytes of accessible data. It is not a file system or real-time data storage system, but rather a long-term storage system for more permanent or static data. Glance provides discovery, registration, and delivery services for virtual disk images.

D. Eucalyptus

Eucalyptus [8] promises the creation of on-premise private clouds, with no requirements for retooling the organization's existing IT infrastructure or need to introduce specialized hardware. Eucalyptus implements an IaaS (Infrastructure as a Service) private cloud that is accessible via an API compatible with Amazon EC2 and Amazon S3. It has five high-level components: Cloud Controller (CLC) that manages the virtualized resources; Cluster Controller (CC) controls the execution of VMs; Walrus is the storage system, Storage Controller (SC) provides block-level network storage including support for Amazon Elastic Block Storage (EBS) semantics; and Node Controller (NC) is installed in each compute node to control VM activities, including the execution, inspection, and termination of VM instances.

IV. Qualitative Feature Comparison of the IaaS Frameworks

All these IaaS frameworks have been designed to allow users to create and manage their own virtual infrastructures. However, these frameworks have differences that need to be considered when choosing a framework. Some qualitative features to consider as part of the selection are summarized in Table 1 and enhance findings from others [9].

Software deployment. An important feature is the ease and frequency of the software deployments. From our own experience the easiest to deploy is OpenNebula because we only have to install a single service in the frontend for a basic configuration while no OpenNebula software is installed in the compute nodes. Nimbus is also relatively easy to install, as only two services have to be configured in the frontend plus the software installation in each compute node. On the other hand, the deployment of Eucalyptus and OpenStack is more difficult due to the number of different components to configure and the different configuration possibilities that they provide. The learning curve for OpenStack is still steep, as the documentation needs some improvements

In addition to a single install we also have to consider update and new release frequencies. From the release notes and announcements of the framework we observe that major updates happen on a four or six months schedule, with many release candidates and upgrades that fix intermediate issues. Furthermore, we observed that the installation deployment depends on scalability requirements. As such, an OpenStack deployment of four nodes may look quite different from one that utilizes 60 nodes. Hence, it is important that integration with configuration management toolkits exist in order to minimize the effort to repeatedly adapt to configuration requirements for a production system. Tools such as chef and puppet provide a considerable value add in this regards. They serve as a repeatable “template” to install the services in case version dependent performance comparisons or feature comparisons are conducted by the users.

Interfaces. Since Amazon EC2 is a de-facto standard, all of them support the basic functionality of this interface, namely image upload and registration, instantiate VMs, as well as describe and terminate operations. However, the OpenStack project noticed disadvantages due to features that EC2 is not exposing. Thus, OpenStack is considering to provide interfaces that diverge from the original EC2.

Storage. Storage is very important in cloud because we have to manage many images and they must be available for users anytime. Therefore, most of the IaaS frameworks decided to provide cloud storage. In the case of Nimbus, it is called Cumulus and it is based on the POSIX filesystem. Cumulus also provides a plugin supporting various storage systems including PVFS, GFS, and HDFS (under a FUSE module). Cumulus uses http/s as a communication protocol.

OpenStack and Eucalyptus provide more sophisticated storage systems. In OpenStack it is called called Swift, and in Eucalyptus it is called Walrus. Both of them are designed to provide fault tolerance and scalability. OpenStack can store the images in either the POSIX filesystem or Swift. In the first case, images are transferred using ssh while in the second one are transferred using http/s. Finally, OpenNebula does not provide a cloud storage product, but its internal storage system can be configured in different ways. Thus, we can have a shared filesystem between frontend and compute nodes; we can transfer the images using ssh; or we can use LVM with CoW to copy the images to the compute nodes.

Networking. The network is managed differently for each IaaS framework while providing various options in each of them:

- Eucalyptus offers four different networking modes: *managed*, *managed-noLAN*, *system*, and *static* [10]. In the two first modes, Eucalyptus manages the network of the VMs. The modes differ in the network isolation provided by vLAN. In the *system* mode, Eucalyptus assumes that IPs are obtained by an external DHCP server. In the static mode, Eucalyptus manages VM IP address assignment by maintaining its own DHCP server with one static entry per VM.
- Nimbus assigns IPs using a DHCP server that can be configured in two ways: centralized and local. In the first case, a DHCP service is used that one configures with Nimbus-specific MAC to IP mappings. In the second case, a DHCP server is installed on every compute node and automatically configured with the appropriate addresses just before a VM boots.
- OpenStack supports two modes of managing networks for virtual machines: flat networking and vLAN networking. vLAN based networking requires a vLAN capable managed switch that you can use to setup vLANs for your systems. Flat Networking uses Linux Ethernet bridging to connect multiple compute hosts together.
- The OpenNebula network contains the following options: host-managed vLANs where the network access is restricted through vLAN tagging, which also requires support from the hardware switches; Ebtables to restrict the network access through Ebtables rules; and Open vSwitch to restrict network access with Open vSwitch Virtual Switch.

Hypervisors. All of the IaaS frameworks support KVM and XEN, which are the most popular open source hypervisors. OpenNebula and Openstack also support VMWare. Eucalyptus only supports VMWare in its

Table 1: Feature comparison of IaaS frameworks. ✓ **indicates a positive evaluation. The more checkmarks the better.**

	OpenStack	Eucalyptus 2.0	Nimbus	OpenNebula
Interfaces	EC2 and S3, Rest Interface. Working on OCCI ✓✓	EC2 and S3, Rest Interface. Working on OCCI ✓✓	EC2 and S3, Rest Interface ✓	Native XML/RPC, EC2 and S3, OCCI, Rest Interface ✓✓✓
Hypervisor	KVM, XEN, VMware Vsphere, LXC, UML and MS HyperV ✓✓✓	KVM and XEN. VMWare in the enterprise edition. ✓✓	KVM and XEN ✓	KVM, XEN and VMWare ✓✓
Networking	- Two modes: (a) Flat networking (b) VLAN networking -Creates Bridges automatically -Uses IP forwarding for public IP -VMs only have private IPs ✓✓✓	- Four modes: (a) managed; (b) managed-novLAN; (c) system; and (d) static - In (a) & (b) bridges are created automatically - IP forwarding for public IP -VMs only have private IPs ✓✓✓	- IP assigned using a DHCP server that can be configured in two ways. - Bridges must exists in the compute nodes ✓✓	- Networks can be defined to support Ebtable, Open vSwitch and 802.1Q tagging -Bridges must exists in the compute nodes -IP are setup inside VM ✓✓✓
Software deployment	- Software is composed by component that can be placed in different machines. - Compute nodes need to install OpenStack software ✓	- Software is composed by component that can be placed in different machines. - Compute nodes need to install OpenStack software ✓	Software is installed in frontend and compute nodes ✓✓	Software is installed in frontend ✓✓✓
DevOps deployment	Chef, Crowbar, Puppet ✓✓✓	Chef*, Puppet* ✓ (*according to vendor)	no	Chef, Puppet ✓✓
Storage (Image Transference)	- Swift (http/s) - Unix filesystem (ssh) ✓	Walrus (http/s) ✓	Cumulus (http/https) ✓	Unix Filesystem (ssh, shared filesystem or LVM with CoW) ✓
Authentication	X509 credentials, LDAP ✓✓✓	X509 credentials ✓	X509 credentials, Grids ✓✓	X509 credential, ssh rsa keypair, password, LDAP ✓✓✓
Avg. Release Frequency	<4month	>4 month ✓	<4 month	>6 month ✓
License	OpenSource - Apache ✓	OpenSource ≠ Commercial	OpenSource Apache ✓	OpenSource Apache ✓

commercial version. Nimbus does not support VMWare. Additionally, OpenStack also supports LXC, UML and Microsoft's HyperV. This makes OpenStack a quite attractive choice for experimenting with different hypervisors environments.

Authentication. All of the IaaS frameworks support X.509 credentials as authentication method for users. OpenStack and OpenNebula also support authentication via LDAP, although it is quite basic. OpenNebula also support ssh rsa keypair and password authentication. OpenStack focuses recently on the development of a keystore. Nimbus allows interfacing with existing Grid infrastructure authentication frameworks.

V. Overview of selected PaaS Frameworks

A. *Platform as a Service*

Platform as a Service (PaaS) offers higher-level services to the users that focus on the delivery of a "platform" to develop and reuse advanced services.. Hence, developers can build applications without installing any tools on their computer and deploy those applications without worrying about system administration tasks. On FutureGrid we provide Hadoop and Twister to our users as they provide popular choices in the community. Twister is a product developed at Indiana University (IU) and used as part of courses taught within the university. This also explains why it is so frequently requested as part of the project registration.

VI. Supported IaaS and PaaS Frameworks in FutureGrid

As already outlined in Section II FutureGrid provides at this time Nimbus, OpenStack, and Eucalyptus on various resources. However, We also experimented with an OpenStack installation with great success. At this time Nimbus is our preferred IaaS framework due to its easy install, the stability, and the support that is provided while including the authors of the Nimbus project as funded partners into the FutureGrid Project. This has a positive impact in support questions, but also in the development of features motivated by FutureGrid. As part of our PaaS offerings, we provide various ways of running Hadoop on FG. This is achieved by either using Hadoop as part of the virtualized environment or exposing it through the queuing system via myHadoop [11]. Twister [12] is contributed through community efforts by IU.

A. *Unicore and Genesis II*

Additionally, services such as Unicore and Genesis II are available as part of the HPC services.

B. Globus

Originally we expected that the demand for Globus [13] would be higher than what we currently see. However, this is explainable due to the following factors: a) Globus demand has been shrinking because many new projects investigate cloud technologies instead of using Grid toolkits; and b) Grid services have been offered in stable form as part of OSG and XSEDE and thus the demand within a testbed is small as sufficient resources are available to experiment with such systems in production environments. Projects that requested Globus include the SAGA team to investigate interoperability, the EMI project, and the Pegasus Project. Nevertheless, only the later has actively requested a real permanent Globus infrastructure on bare metal resources. The other projects have indicated that their needs can be fulfilled by *raining* Globus onto a server based on images that are managed by the projects themselves. Furthermore, we observe that the Globus project has moved over the last year to deliver package based installs that make such deployments easy. We had only one additional user that requested not the installation of Globus, but the installation of GridFTP to more easily move files along. This request could be satisfied while using the CoG Kit [14] and abstracting the need for moving data away from a protocol dependency. Interoperability projects would be entirely served by a virtual Globus installation on images via rain (see Section VII). In many of these cases even bare-metal access is not needed. In fact such use cases benefit from the ability to start up many Globus services to as part of the development of advanced workflows that are researched by the community.

At this time we are not supporting any other PaaS such as messaging queues or hosted databases. However, we intend to adapt our strategies based on evaluating user requests with available support capabilities.

VII. RAINING

Due to the variety of services and limited resources provided in FG, it is necessary to enable a mechanism to provision needed services onto resources. This includes also the assignment of resources to different IaaS or PaaS frameworks. We have developed, as first step to address this challenge, a sophisticated image management toolkit that allows us to not only provision virtual machines, but also provision directly onto bare-metal. Thus, we use the term *raining* to indicate that we can place arbitrary software stack onto a resource. The toolkit to do so is called *rain*.

Rain makes it possible to compare the benefits of IaaS, PaaS performance issues, as well as evaluating which applications can benefit from such environments and how they must be efficiently configured. As part of this process, we allow the generation of abstract images and universal image registration with the various infrastructures including Nimbus, Eucalyptus, OpenNebula, OpenStack, but also bare-metal via the HPC services. This complex process is described in more detail in [15]. It is one of the unique features about FutureGrid to provide an essential component to make comparisons between the different infrastructures more easily possible. Our toolkit rain is tasked with simplifying the full life cycle of the image management process in FG. It involves the process of creating, customizing, storing, sharing and deploying images for different FG environments. One important feature in our image management design is that we are not simply storing an image but rather focus on the way an image is created through templating. Thus it will be possible at any time to regenerate an image based on the template that is used to install the software stack onto a bare operating system. In this way, we can optimize the use of the storage resources.

Now that we have an elementary design of managing images, we can dynamically provision them to create bare-metal and virtualized environments while raining them onto our resources. Hence, rain will offers four main features:

- Create customized environments on demand.
- Compare different infrastructures.
- Move resources from one infrastructure to another by changing the image they are running plus doing needed changes in the framework.
- Ease the system administrator burden for creating deployable images.

This basic functionality has been tested as part of a scalability experiment motivated by our use case introduced in the introduction. Here, a user likes to start many virtual machines and compare the performance of this task between the different frameworks.

VIII. INFRASTRUCTURE SCALABILITY STUDY

We have performed several tests to study the scalability of the infrastructures installed in our cluster at Indiana University called India. The India cluster is composed by Intel Xeon X5570 with 24GB of memory, a single drive of 500GB with 7200RPMm 3Gb/s and an interconnection network of 1Gbps Ethernet.

The idea of these tests is to provision as many physical machines (PM) or virtual machines (VM) at the same time as possible. Obviously this is an important test that is of especial interest to our traditional HPC community that needs many compute resources to operate in parallel. Tests succeed if all the machines have ssh access and we can successfully connect and run commands on all the machines. We measure the time that takes since the request is first issued until we have access to all the machines.

For that purpose we have used our image generator and registration tools and created a CentOS 5 image. We registered it in the different infrastructures: HPC, OpenStack, Eucalyptus and OpenNebula. This process gives us an identical image template. Therefore, the only difference in the image is the version of the 2.6 kernel/ramdisk used. In HPC we use the ramdisk modified by xCAT, in Eucalyptus we use a XEN kernel and in OpenStack or OpenNebula we use a generic kernel with KVM support. The total size of the image without compression is 1.5 GB. In the case of netboot it is compressed to about 300MB. Figure 5 shows the results of the performed tests. In the following sections we describe the software used in each infrastructure, the results obtained and the problems we found.

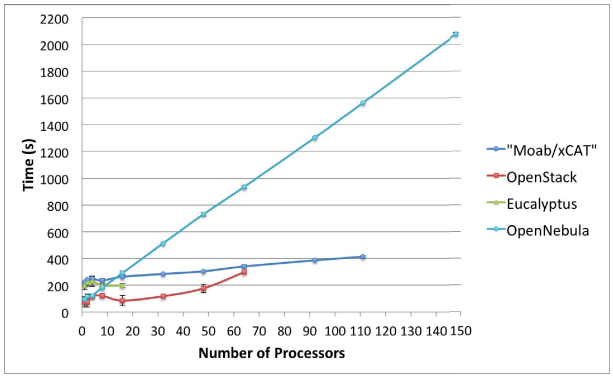

Figure 5: Scalability Experiment of IaaS Frameworks on FutureGrid

A. *HPC (Moab/xCAT)*

We used Moab 6.0.3 and xCAT 2.6.9 to dynamically provision images to the compute nodes. We had a total of 111 nodes provisioned with the same image. During the tests we did not have any problems and observed very good linear scalability. This behavior is based on the good behavior of execution the provisioning step in parallel. A small bottleneck in our experiments was introduced at the time of retrieving the image from the server. Since the image is compressed and relatively small, the scalability of this setup is preserved. However, provisioning even a single image takes more time in comparison to its virtualized counterparts. This easily explained by the fact that the process of rebooting the physically machine must be called which is naturally not needed in case of using virtualized machines. Here the process of rebooting takes place in memory and is less time consuming.

B. *OpenStack*

In our experiments, we have used the Cactus version of OpenStack. In order to make an efficient image provisioning, OpenStack caches images in the computes nodes. Thus, it does not need to transfer the image over the network every time it is requested and the instantiation of a VM is faster.

Due to scalability issues by provisioning higher number of VMs, we modified our strategy to submit VM requests in batches of 10 VMs at a time (maximum). This means that in the case of provisioning 32 VMs, we have requested 3 times 10 VMs and once 2 VMs. This is a verified bug in OpenStack and is going to be solved in future releases. Or through the method we chose for Cactus [16].

Furthermore, we observed that if the image is not cached in the compute nodes, the scalability of this framework is very limited. In fact, we started to have problems trying to boot 16 VMs and observed a failure rate of about 50% for our test. We observed that VMs got *stuck* in the launching status (this is the status where the image is transferred to the compute node).

On the other hand, once we had the image cached in most of the compute nodes, we were able to boot up to 64 VMs simultaneously. However, placing the images into the cache was not an easy task, as we had to deal with a 50% failure rate. Besides observing that the image got stuck in the launching status, we also observed that other images were running, but we could not access them via ssh. The later problem is mainly related to the fact that OpenStack does not inject the public network configuration inside the VM. Alternatively, it creates bridges and conducts IP forwarding to direct all the traffic toward the private IP. At times we observed that the bridges, the iptables entries or both were not created properly. In particular, the iptables issue is important because we observed that sometimes OpenStack gets confused and duplicate entries in the iptables using old information that was not properly cleaned up. This erroneous information makes the VMs inaccessible. We observed that this problem is persistent and one needs to manually modify the database to remove the wrong entries or restart OpenStack.

Finally, another known problem of OpenStack Cactus is that it is not able to automatically assign public IPs to the VMs, so we have to look into the pool and assign one to each VM manually. This feature has been added in the Diablo version, though. Our intention is to rerun our experiments also with Diablo once it is available on FutureGrid.

C. *Eucalyptus*

We use version 2.03 of Eucalyptus, which is the latest open source version. Eucalyptus also caches the images in the compute nodes. Just like in OpenStack, we observed similar issues with Eucalyptus while requesting larger numbers of VMs. Thus, we modified our initial experiment plan and also introduced batched requests (10 at a time) but with a delay of 6 seconds. This is due to an annoying feature of Eucalyptus that prevents users from executing the same command several times in a short period of time.

The tests with this infrastructure were quite disappointing because we could only get 16 VMs running at the same time. Even though, the failure rate was very high. Eucalyptus, like OpenStack, configures the public network with IP forwarding and creates the bridges at running time. Thus, this creates problems, like in the case of OpenStack, we got similar errors like missing bridges and iptables entries. Additionally, we had problems with the automatic assignment of public IPs to the VMs. This means that some VMs did not get public IPs and therefore they were not accessible for users.

D. *OpenNebula*

We used OpenNebula version 3.0.0. By default OpenNebula does not cache images in the compute nodes. It supports three basic transfer plugins named nfs, ssh and lvm. NFS has a terrible performance because the VMs are reading the image through the network. SSH was the one that we used because is still easy to configure and has a better performance. The last one seems more difficult to configure, but it should provide the best performance, because it is the one selected by the OpenNebula team to perform their experiments [17, 18]. We were able to instantiate 148 VMs at the same time with almost a 100% of success. In fact, we only got one error in the case of 148 VMs. Since our configuration does not use an image cache, it was very slow

and limited by the network, as each image must be copied to each VM. This is done in a sequential process. However, this can be improved by introducing caches as others have showcased [19].

IX. Conclusions

This paper provides evidence that offering not one but many IaaS frameworks is needed to allow users to answer the question which IaaS framework is right for them. We have to be aware that users requirements may be very different from each other and the answer to this question cannot be universally cast.

We identified an experiment to be executed on FutureGrid to provide us with feedback on a question many of our users have: Which infrastructure framework should I use under the assumption I have to start many instances. Our close ties to the HPC community and researchers utilizing parallel services and programming methodologies naturally motivate this experiment. However, such users tend not to be experts in IaaS frameworks and a tool is needed to allow the comparison of their use cases in the different frameworks. In this paper, we have demonstrated that we achieved a major milestone towards this goal and proven that our efforts can lead to significant comparisons between deployed frameworks. This also includes the ability to identify a set of benchmarks for such infrastructures in order to further identify which framework is best suited for a particular application. Our test case must be part of such a benchmark.

Concretely, we found challenges in our scalability experiments while raining images on them. This was especially evident for Eucalyptus and even OpenStack. As many components are involved in the deployment they are also not that easy to deploy. Tools provided as part of developments such as Chef [20] and Puppet [21] can simplify deployments especially if they have to be done repeatedly or require modifications to improve scalability. We claim that the environment to conduct an OpenStack experiment with just a handful VMs may look quite different from a deployment that uses many hundreds of servers on which Openstack may be hosted. This is also documented nicely as part of the OpenStack Web pages that recommends more complex service hierarchies in case of larger deployments.

On the other hand, we have seen that OpenNebula is very reliable and easy to deploy. Although in our experiments we found it is quite slow due to the lack of caching images. Nevertheless, we think that this problem is easy to solve. In fact, after we finished the tests we also found other community members having the same problem. A plugin to provide caching support is already available, which can significantly improve managing images across many nodes in OpenNebula.

Through the ability of rain it will become easier for us to deploy PaaS on the IaaS offerings, as we will be able to create “templates” that facilitate their installation and potentially their upgrade. Due to this ability it is possible to replicate the environments and introduce reproducible environment.

The answer to the question, which IaaS question should one use, depends on the requirements of the applications and the user community. OpenNebula and Nimbus were easier to install, but we observed that OpenStack and Eucalyptus have considerably worked on these issues in newer versions to be released shortly. Nimbus has turned out to be very reliable, but our OpenStack and Eucalyptus Clouds have been extremely popular resulting in resource starvation. We will address this issue by assigning dynamically resources between the clouds. From our own observations working with applications it is clear that we do need a testbed, such as FutureGrid, to allow users to cast their own evaluations.

Acknowledgements

We would like to acknowledge the rest of the FG team for its support in the preparation and execution of these tests. This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

References

[1] *Nimbus Project Web Page*. http://www.nimbusproject.org, May 2012.
[2] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid," *Scientific Programming Journal,* vol. 13, pp. 265-276, 2005.
[3] *OpenNebula Web Page*. http://www.opennebula.org, May 2012.
[4] I. M. Llorente, R. Moreno-Vozmediano, and R. S. Montero, "Cloud computing for on-demand grid resource provisioning," *Advances in Parallel Computing,* vol. 18, pp. 177-191, 2009.
[5] R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "An Elasticity Model for High Throughput Computing Clusters," *J. Parallel and Distributed Computing,* 2010.
[6] B.Sotomayor, R. S. Montero, I. M. Llorente, and I.Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing,* vol. 13, pp. 14-22, Sep.-Oct 2009 2009.
[7] *OpenStack Web Page*. http://www.openstack.org, May 2012.
[8] *Eucalyptus Web Pages (Open Source)*. http://open.eucalyptus.com/, May 2012.
[9] P. Sempolinski and D. Thain, " A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. ," presented at the IEEE 2nd Int. Conf. on Cloud Computing Technology and Science, 2010.
[10] *Eucalyptus Network Configuration 2.0*. http://open.eucalyptus.com/wiki/EucalyptusNetworkConfiguration_v2.0, May 2012.
[11] *myHadoop*. http://sourceforge.net/projects/myhadoop/, May 2012.
[12] *Twister*. http://www.iterativemapreduce.org/, May 2012.
[13] Globus Project. http://www.globus.org/, May 2012.
[14] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience,* vol. 13, pp. 643-662, 2001.
[15] J. Diaz, G. v. Laszewski, F. Wang, and G. Fox, " Abstract Image Management and Universal Image Registration for Cloud and HPC Infrastructures " IEEE Cloud. 2012.
[16] Running 200 VM instances on OpenStack http://blog.griddynamics.com/2011/05/running-200-vm-instances-on-openstack.html, May 2012.
[17] U. Schwickerath, "CERN Cloud Computing Infrastructure," presented at the ICS Cloud Computing Conference, 2010.
[18] CERN Scaling up to 16000 VMs. http://blog.opennebula.org/?p=620, May 2012.
[19] Cache optimization in OpenNebula. http://dev.opennebula.org/issues/553, May 2012.
[20] *Chef*. http://www.opscode.com/chef/, May 2012
[21] Puppet. http://puppetlabs.com/, May 2012.