Cloud Gazing: A Comprehensive Evaluation of IaaS Technologies for FutureGrid

Andrew J. Younge, FutureGrid Software Team
Pervasive Technology Institute, Indiana University
2729 E 10th St., Bloomington, IN 47408, U.S.A.
Email corresponding author: ajyounge@indiana.edu

Abstract—Since the beginning of Cloud computing as a new and emerging paradigm, the field of Distributed Systems is amidst a stage of redefinition. Due to this, there are a wide variety of "Cloud" solutions, ranging from virtualization technologies to internet enabled web services, however a Cloud usually entails the use of Infrastructure as a Service for many people. Lately, there have been a number of private Infrastructure platforms with little formal evaluation of each. As such, this manuscript looks to evaluate various Infrastructure level cloud technologies, specifically for supporting advanced scientific research. The FutureGrid project, a high performance cloud and grid testbed, provides the ideal environment for an unbiased evaluation.

I. INTRODUCTION

Cloud computing [1] is one of the most explosively expanding technologies in the computing industry today. A Cloud computing implementation typically enables users to migrate their data and computation to a remote location with some varying impact on system performance [2]. This provides a number of benefits which could not otherwise be achieved. Such benefits include:

- **Scalability** - Clouds are designed to deliver as much computing power as any user needs. While in practice the underlying infrastructure is not infinite, the cloud resources are projected to ease the developer's dependence on any specific hardware.

- **Quality of Service (QoS)** - Unlike standard data centers and advanced computing resources, a well-designed Cloud can project a much higher QoS than traditionally possible. This is due to the lack of dependence on specific hardware, so any physical machine failures can be mitigated without the prerequisite user awareness.

- **Customization** - Within a Cloud, the user can utilize customized tools and services to meet their needs. This can be to utilize the latest library, toolkit, or to support legacy code within new infrastructure.

- **Cost Effectiveness** - Users find only the hardware required for each project. This reduces the risk for institutions potentially want to build a scalable system, thus providing greater flexibility, since the user is only paying for needed infrastructure while maintaining the option to increase services as needed in the future.

- **Simplified Access Interfaces** - Whether using a specific application, a set of tools or Web services, Clouds provide access to a potentially vast amount of computing resources in an easy and user-centric way.

While Cloud computing has been driven from the start predominantly by the industry through Amazon [3], Google [4] and Microsoft [5], a shift is also occurring within the academic setting as well. Due to the many benefits, Cloud computing is becoming immersed in the area of High Performance Computing (HPC), specifically with the deployment of scientific clouds [6] and virtualized clusters [7]. Unlike most users of public clouds, scientists typically have a much larger demand for computing power, so public cloud deployments, especially at the infrastructure level, aren't practical. This allows the scientific community to leverage one of the key aspects of Cloud computing; scalability through economies of scale [8]. The fundamental concept is to provide a unified Infrastructure-as-a-Service deployments to a wide variety of scientific users ranging from astronomy, chemistry, biology, engineering, atmospheric science and epidemiology, thereby pooling resources and providing a near-infinite view of computing power, metered as a utility. One of the major deliverables of the FutureGrid project [9] is to provide such a service and more to the scientific community. However, in order to accomplish this, its imperative to make a detailed and accurate analysis of current infrastructure level cloud deployments, and evaluate what advantages shortcomings exist in our path to provide this scientific Cloud at a national scale.

The rest of this paper is organized as follows. First, we discuss the variation that exists within the Cloud community, what public Cloud offerings exist, and the FutureGrid project itself. Next, we look at the most prominent IaaS deployments that have potential for use within FutureGrid, and what immediate advantages and disadvantages exist. Then, we discuss the deployment and implementation of each process, and conclude with the insights as to each service's potential use at a national scale within FutureGrid.

II. RELATED WORK WITHIN CLOUDS

Cloud computing is one of the most explosively expanding technologies in the computing industry today. However it is important to understand where it came from, in order to figure out where it will be heading in the future. While there is no clear cut evolutionary path to Clouds, many believe the concepts originate from two specific areas: Grid Computing and Web 2.0.

Grid computing [10], [11], in its practical form, represents the concept of connecting two or more spatially and administratively diverse clusters or supercomputers together in a
federating manner. The term “the Grid” was coined in the mid 1990’s to represent a large distributed systems infrastructure for advanced scientific and engineering computing problems. Grids aim to enable applications to harness the full potential of resources through coordinated and controlled resource sharing by scalable virtual organizations. While not all of these concepts carry over to the Cloud, the control, federation, and dynamic sharing of resources is conceptually the same as in the Grid. This is outlined by [12], as Grids and Clouds are compared at an abstract level and many concepts are remarkably similar. From a scientific perspective, the goals of Clouds and Grids are also similar. Both systems attempt to provide large amounts of computing power by leveraging a multitude of sites running diverse applications concurrently in symphony. The only significant differences between Grids and Clouds exist in the implementation details, and the reproductions of them, as outlined later in this section.

The other major component, Web 2.0, is also a relatively new concept in the history of Computer Science. The term Web 2.0 was originally coined in 1999 in a futuristic prediction by Dracy DiNucci [13]: “The Web we know now, which loads into a browser window in essentially static screenfulls, is only an embryo of the Web to come. The first glimmerings of Web 2.0 are beginning to appear, and we are just starting to see how that embryo might develop. The Web will be understood not as screenfulls of text and graphics but as a transport mechanism, the ether through which interactivity happens. It will [...] appear on your computer screen, [...] on your TV set [...] your car dashboard [...] your cell phone [...] hand-held game machines [...] maybe even your microwave oven.” Her vision began to form, as illustrated in 2004 by the O’Riley Web 2.0 conference, and since then the term has been a pivotal buzz word among the internet. While many definitions have been provided, Web 2.0 really represents the transition from static HTML to harnessing the Internet and the Web as a platform in of itself.

Web 2.0 provides multiple levels of application services to users across the Internet. In essence, the web becomes an application suite for users. Data is outsourced to wherever it is wanted, and the users have total control over what they interact with, and spread accordingly. This requires extensive, dynamic and scalable hosting resources for these applications. This demand provides the user-base for much of the commercial Cloud computing industry today. Web 2.0 software requires abstracted resources to be allocated and relinquished on the fly, depending on the Web’s traffic and service usage at each site. Furthermore, Web 2.0 brought Web Services standards [14] and the Service Oriented Architecture (SOA) [15] which outline the interaction between users and cyberinfrastructure. In summary, Web 2.0 defined the interaction standards and user base, and Grid computing defined the underlying infrastructure capabilities.

Cloud computing in its physical form has many different meanings and forms. Since Clouds are defined by the services they provide and not by applications, an integrated as-a-service paradigm has been defined to illustrate the various levels within a typical Cloud, as in Figure 1.

- **Clients** - A client interacts with a Cloud through a predefined, thin layer of abstraction. This layer is responsible for communicating the user requests and displaying data returned in a way that is simple and intuitive for the user. Examples include a Web Browser or a thin client application.
- **Software-as-a-Service (SaaS)** - A framework for providing applications or software deployed on the Internet packaged as a unique service for users to consume. By doing so, the burden of running a local application directly on the client’s machine is removed. Instead all the application logic and data is managed centrally and to the user through a browser or thin client. Examples include Google Docs, Facebook, or Pandora.
- **Platform-as-a-Service (PaaS)** - A framework for providing a unique computing platform or software stack for applications and services to be developed on. The goal of PaaS is to alleviate many of the burdens of developing complex, scalable software by proving a programming paradigm and tools that make service development and integration a tractable task for many. Examples include Microsoft Azure and Google App Engine.
- **Infrastructure-as-a-Service (IaaS)** - A framework for providing entire computing resources through a service. This typically represents virtualized Operating Systems, thereby masking the underlying complexity details of the physical infrastructure. This allows users to rent or buy computing resources on demand for their own use without needing to operate or manage physical infrastructure. Examples include Amazon EC2, Eucalyptus, and Nimbus.
- **Physical Hardware** - The underlying set of physical machines and IT equipment that host the various levels of service. These are typically managed at a large scale using virtualization technologies which provide the QoS users expect. This is the basis for all computing infrastructure.

When all of these layers are combined, a dynamic software stack is created to focus on large scale deployment of services to users. However, this paper looks to concentrate on a key level, the IaaS deployment. This is because it is the most rootametary of all all Cloud deployments, as all Platform and Software services are dependent on its completeness. Further-
more, its the most applicable to the widest array of scientific researchers looking to leverage Cloud technologies.

A. FutureGrid

FutureGrid is a national-scale Grid and Cloud test-bed facility that includes a number of computational resources across many distributed locations. The FutureGrid network is unique and can lend itself to a multitude of experiments specifically for evaluating middleware technologies and experiment management services. This network can be dedicated to conduct experiments in isolation, using a network impairment device for introducing a variety of predetermined network conditions. Figure 2 depicts the geographically distributed resources that are outlined in Table I in more detail. All network links within FutureGrid are dedicated 10GbE links with the exception of a shared 10GbE link to TACC over the TeraGrid [16], [17] network, enabling high-speed data management and transfer between each partner site within FutureGrid.

![FutureGrid Participants and Resources](image)

Although the total number of systems within FutureGrid is comparatively conservative, they provide some heterogeneity to the architecture and are connected by the high-bandwidth network links. One important feature to note is that most systems can be dynamically provisioned, e.g. these systems can be reconfigured when needed by special software that is part of FutureGrid with proper access control by users and administrators. Therefore its believed that this hardware infrastructure can fully accommodate the needs of an experiment management system.

III. IaaS Overview

In order to properly evaluate each IaaS technology, a detailed overview to each major technology is necessary. This includes Nimbus, Eucalyptus, OpenStack, and Open Nebula; the four major open IaaS deployments currently available to the broader community.

A. Nimbus

Nimbus [18], [19] is a set of open source tools that together provide an Infrastructure as a Service (IaaS) cloud computing solution, see Figure 3. It is based on the concept of virtual workspaces previously introduced for Globus [19]. A virtual workspace is an abstraction of an execution environment that can be made dynamically available to authorized clients by using well-defined protocols. In this way, it can create customized environments by deploying virtual machines (VMs) among remote resources. To such an end, Nimbus provides a web interface called Nimbus Web. Its aim is to provide administrative and user functions in a friendly interface. Nimbus Web is centered around a Python Django [20] web application that is intended to be deployable completely separate from the Nimbus service.

![Nimbus Cloud Infrastructure](image)

As we can see in Figure 3, a storage cloud implementation called Cumulus [18] has been tightly integrated with the other central services, although it can also be used standalone. Cumulus is compatible with the Amazon Web Services S3 REST API [21], but extends its capabilities by including features such as quota management. The Nimbus cloud client uses the Jets3t library [22] to interact with Cumulus. However, since it is compatible with S3 REST API, we could use other interfaces like boto [23] or s2cmd [24].

Nimbus supports two resource management strategies. The first one is the default “resource pool” mode. In this mode, the service has direct control of a pool of virtual machine managers (VMM) nodes and it assumes it can start VMs. The other supported mode is called “pilot”. Here, the service makes requests, to a cluster’s Local Resource Management System (LRMS), to get a VMM available where deploy VMs.

Nimbus also provides an implementation of Amazon’s Elastic Compute Cloud (EC2) [25] that allows you to use clients developed for the real EC2 system against Nimbus based clouds.

B. Eucalyptus

Eucalyptus is a product from Eucalyptus Systems [26], [27], that developed out of a research project at the University of California, Santa Barbara. Eucalyptus was initially aimed at bringing the cloud computing paradigm of computing to academic super computers and clusters. Eucalyptus provides a Amazon Web Services (AWS) complaint EC2 based web service interface for interacting with the Cloud service. Additionally Eucalyptus provides services as well, such as the AWS Complaint Walrus and a user interface for managing users and images.
TABLE I
FUTUREGRID HARDWARE

<table>
<thead>
<tr>
<th>System type</th>
<th># CPUs</th>
<th># Cores</th>
<th>TFLOPS</th>
<th>RAM (GB)</th>
<th>Storage (TB)</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM iDataPlex</td>
<td>256</td>
<td>1024</td>
<td>11</td>
<td>3072</td>
<td>1.335</td>
<td>IU</td>
</tr>
<tr>
<td>Dell PowerEdge</td>
<td>192</td>
<td>1152</td>
<td>12</td>
<td>1152</td>
<td>15</td>
<td>TACC</td>
</tr>
<tr>
<td>IBM iDataPlex</td>
<td>168</td>
<td>672</td>
<td>7</td>
<td>2016</td>
<td>120</td>
<td>UC</td>
</tr>
<tr>
<td>IBM iDataPlex</td>
<td>168</td>
<td>672</td>
<td>7</td>
<td>2688</td>
<td>72</td>
<td>UCSD</td>
</tr>
<tr>
<td>Cray XT5m</td>
<td>168</td>
<td>672</td>
<td>6</td>
<td>1344</td>
<td>1.335</td>
<td>IU</td>
</tr>
<tr>
<td>Shared mem. sys.</td>
<td>40</td>
<td>480</td>
<td>4</td>
<td>640</td>
<td>1.335</td>
<td>IU</td>
</tr>
<tr>
<td>IBM iDataPlex</td>
<td>64</td>
<td>256</td>
<td>2</td>
<td>768</td>
<td>5</td>
<td>UF</td>
</tr>
<tr>
<td>Total</td>
<td>1337</td>
<td>5600</td>
<td>58</td>
<td>10560</td>
<td>552</td>
<td></td>
</tr>
</tbody>
</table>

1Indicates shared file system. 2Best current estimate

General Architecture. The Eucalyptus system architecture as presented in the Eucalyptus white paper [28] is shown below: The architecture is based on a two level hierarchy of the Cloud controller and the Cluster controller. The cluster controller usually manages the nodes within a single cluster and multiple such Cluster Controllers can be used to connect to a single Cloud Controller. The Cloud controller is responsible for the resource management, scheduling and accounting aspects of the Cloud.

Security. Eucalyptus being AWS complaint provides the same form of authentication that AWS supports, namely the shared key and PKI models. The shared keys are used with the Eucalyptus Query Interface.

User Interface. Eucalyptus, being one of the first private cloud computing solutions, has a very well focused user interface. Much of Eucalyptus’s design is based on the functionality of Amazon’s EC2 cloud solution, and the user interface is a prime example. While EC2 is a proprietary public cloud, it uses an open interface through the use of well designed Web Services which are open to all. Eucalyptus, looking to provide complete compatibility with EC2 to market the private cloud market, uses the same interface for all communication to the Cloud Controller (CLC).

While Eucalyptus can be controlled using the EC2 AMI tools, it also provides its own specific tool set; euca2ools. Euca2ools provides support for creating and managing key-pairs, querying the cloud system, managing VMs, starting and terminating instances, network configuration, and block storage usage. The Eucalyptus system also provides a secure web front end to allow new users to create and manage account information, view available VMs, and download their security credentials.

Image Management. As seen with the user interface, Eucalyptus takes many design queues from Amazons EC2 and the Image management system is no different. Eucalyptus stores images in Walrus, the block storage system that is analogous to the Amazon S3 service. As such, any user can bundle there own root filesystem, upload and then register this image and link that image with a particular kernel and ramdisk image. This image is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime from any availability zone. This allows users to create specialty virtual appliances and deploy them within Eucalyptus with ease.

C. OpenStack

Open Stack has been introduced by Rackspace and NASA in July 2010. The project is trying to build an open source community spanning technologists, developers, researchers, and industry to share resources and technologies with the goal to create a massively scalable and secure cloud infrastructure. In tradition with other open source projects the entire software is open sources and limited to just open source API’s such as Amazon.

Currently, OpenStack focuses on the development of two aspects of cloud computing to address compute and storage aspects with their OpenStack Compute and OpenStack Sorage solutions. According to the documentation “OpenStack Compute is the internal fabric of the cloud creating and managing large groups of virtual private servers” and “OpenStack Object Storage is software for creating redundant, scalable object storage using clusters of commodity servers to store Terabytes or even petabytes of data.” Recently, an image repository has been prototyped. The image repository contains an image registration and discovery service and an image delivery service. Together they deliver images to the compute service while obtaining them from the storage service. This development gives an indication that the project is striving to integrate more services into their portfolio.

OpenStack Compute. As part of its computing support efforts OpenStack [29] is developing a cloud computing fabric controller, a component of an IaaS system, known under the name Nova. The architecture for Nova is built on the concepts of shared-nothing and messaging-based information exchange. Hence most communication in Nova are facilitated by message queues. To prevent blocking components while waiting for a response from others, deferred objects are introduced. Such objects include callback that gets triggered when a response
is received. This is very similar to established concepts from parallel computing such as “futures” which have been successfully utilized in the Grid community by projects such as the CoG Kit.

To achieve the shared-nothing paradigm, the overall system state is kept in a distributed data system. State updates are made consistent through atomic transactions. Nova is implemented in python while utilizing a number of externally supported libraries, and components. This includes boto an Amazon API provided in python [23], and Tornado a fast HTTP server used to implement the S3 capabilities in OpenStack. Figure OS.1 shows the main architecture of Open Stack Compute. In this architecture the API Server receives http requests from boto, converts the commands to and from the API format while forwarding requests to the cloud controller. The cloud controller maintains the global state of system, assures authorization while interacting with the User Manager via LDAP, interacts with the S3 service and manages nodes, as well as storage workers through a queue.

Additionally Nova integrates networking components to manage private networks, public IP addressing, VPN connectivity, and firewall rules. They include the following types:

- NetworkController, that manages address and vlan allocations,
- RoutingNode, that governs the NATs public IPs to private IPs, and enforces firewall rules,
- AddressingNode, that runs DHCP services for private networks, and
- TunnelingNode, that provides VPN connectivity.

The network state (managed in the distributed object store) allows the following:

- VLAN assignment to a project,
- Private Subnet assignment to a security group in a VLAN,
- Private IP assignments to running instances,
- Public IP allocations to a project, and
- Public IP associations to a private IP / running instance.

OpenStack Storage. The OpenStack storage solution is build around a number of interacting components and concepts including a Proxy Server, a Ring, Object Server, a Container Server, an Account Server, Replication, Updaters, and Auditors.

The role of the Proxy Server is to enable of look ups to the location of the accounts, containers, or objects in OpenStack storage rings and route the request. Thus any object is streamed to or from an object server directly through the proxy server to or from the user. A ring represents a mapping between the names of entities stored on disk and their physical location. Separate rings for accounts, containers, and objects exist. A ring includes the concepts of using zones, devices, partitions, and replicas. Hence it allows dealing with failures, and isolation of zones representing a drive, a server, a cabinet, a switch, or even a datacenter. Weights can be used to balance the distribution of partitions on drives across the cluster allowing to support heterogeneous storage resources. According to the documentation, “the Object Server is a very simple blob storage server that can store, retrieve and delete objects stored on local devices.” Objects are stored as binary files with metadata stored in the file’s extended attributes. This requires that the underlying filesystem choice for object servers support which is often not the case for standard Linux installations. To list objects, a Container Server can be utilized. Listing of containers is handled by the Account Server.

At this time the documentation of OpenStack indicates that the software is not yet ready for production services. The project has achieved a significant amount of publication and support. However the documentation of the project has at this time just started and is improved by its partners.

D. OpenNebula

OpenNebula [30], [31] is an open-source toolkit which allows to transform existing infrastructure into an Infrastructure as a Service (IaaS) cloud with cloud-like interfaces. Figure 4 shows the OpenNebula architecture and their main components.

```

<table>
<thead>
<tr>
<th>Web Interface</th>
<th>CLI</th>
<th>Other Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenNebula Core</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drivers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtualization</td>
<td>Network</td>
<td>Storage</td>
</tr>
<tr>
<td>Local Infrastructure</td>
<td>External Cloud</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 4. OpenNebula Architecture
```

The architecture of OpenNebula has been designed to be flexible and modular to allow its integration with different storage and network infrastructure configurations, and hypervisor technologies. Here, the core is a centralized component that manage the virtual machine’s (VM) full life cycle, including setting up networks dynamically for groups of VMs and managing their storage requirements, such as VM disk image deployment or on-the-fly software environment creation. Other important component is the capacity manager or scheduler. It governs the functionality provided by the core. The default capacity scheduler is a requirement/rank matchmaker. However, it is also possible to develop more complex scheduling policies, through a lease model and advance reservations like Haizea [32]. The last main components are the access drivers. They provide an abstraction of the underlying infrastructure to expose the basic functionality of the monitoring, storage and virtualization services available in the cluster. Therefore, OpenNebula is not tied to any specific environment and can provide a uniform management layer regardless of the virtualization platform.

Additionally, OpenNebula offers management interfaces to integrate the core’s functionality within other data center management tools, such as accounting or monitoring frameworks. To this end, OpenNebula implements the libvirt API [33], an
open interface for VM management, as well as a command line interface (CLI). A subset of this functionality is exposed to external users through a cloud interface.

Due to its architecture, OpenNebula is able to adapt to organizations with changing resource needs, including the addition or failure of physical resources [34]. Some essential features to support changing environments are the live migration and the snapshotting of VMs [30]. Furthermore, when the local resources are insufficient, OpenNebula can support a hybrid cloud model by using cloud drivers to interface with external clouds. This lets organizations supplement the local infrastructure with computing capacity from a public cloud to meet peak demands, or implement high availability strategies. OpenNebula currently includes an EC2 driver, which can submit requests to Amazon EC2 [25] and Eucalyptus [26], as well as an ElasticHosts driver [35].

Regarding the storage, an Image Repository allows users to easily specify disk images from a catalog without worrying about low-level disk configuration attributes or block device mapping. Also, image access control is applied to the images registered in the repository, hence simplifying multi-user environments and image sharing. Nevertheless, users can also set up their own images.

IV. Evaluation and Review

This section will provide a much more analytical review for each Cloud technology. Specific comparisons will be made between each service from a capabilities, user, administrative, and security aspects. Expect detailed charts in regards to feature support and requirements.

V. Deployment & Experience

This section will detail the experimental analysis of each major IaaS deployment. It will discuss specifics regarding installation, administrative management, and user experiences. Specifics regarding user use cases, the user interface choices, specific usage numbers, etc will all be provided. From the administrative side, the advantages and difficulties experienced will provide a proper review of IaaS deployments from the project management side. These two views should properly illustrate the ability of each IaaS deployment.

VI. Conclusion

TBD

ACKNOWLEDGMENT

This material is based on the support from the National Science Foundation (NSF) under Grant No. 0910812 to Indiana University for FutureGrid: An Experimental, High-Performance Grid Test-bed. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

REFERENCES


