A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus

Peter Sempolinski and Douglas Thain
University of Notre Dame

Abstract

Eucalyptus, OpenNebula and Nimbus are three major open-source cloud-computing software platforms. The overall function of these systems is to manage the provisioning of virtual machines for a cloud providing infrastructure-as-a-service. These various open-source projects provide an important alternative for those who do not wish to use a commercially provided cloud.

We provide a comparison and analysis of each of these systems. We begin with a short summary comparing the current raw feature set of these projects. After that, we deepen our analysis by describing how these cloud management frameworks relate to the many other software components required to create a functioning cloud computing system. We also analyse the overall structure of each of these projects and address how the differing features and implementations reflect the different goals of each of these projects. Lastly, we discuss some of the common challenges that emerge in setting up any of these frameworks and suggest avenues of further research and development. These include the problem of fair scheduling in absence of money, eviction or preemption, the difficulties of network configuration, and the frequent lack of clean abstractions.

1 Introduction

In recent years, due to the high demand for computing resources for processing jobs that require hundreds or even thousands of cores, different paradigms have been developed for harnessing the computational power of large groups of processors. Depending on the computational setting and the problem being solved, different approaches are being taken, including batch job systems such as SGE [11], cycle scavenging with Condor [27], or virtual machine spawning with Amazon's EC2 cloud [2].

The various definitions of cloud, grid or distributed computing are open to some debate. [29] We will confine ourselves to considering the idea of infrastructure-as-a-service, accomplished by providing virtual machines to users. Amazon's EC2 cloud is arguably one of the best examples of this paradigm. [2] Of course, this is not to minimize vast varieties of system configurations that can be referred to as "clouds". We also point out that Amazon is not the only player in cloud computing market, as Sun (Sun Cloud), IBM (Blue Cloud), Microsoft (Azure), and many others have their own systems as well. [7] [13] [21]

In the setting we are considering, a cloud is a group of machines configured in such a way that an end-user can request any number of virtual machines (VMs) of a desired configuration. The cloud will spawn these VMs somewhere on the physical machines that it owns. The word "cloud" in this context is meant to convey the semi-ethereal nature of these VMs. The end-user neither knows nor cares where exactly these VMs are physically located or the configuration of the underlying hardware, so long as they can access their bank of properly configured VMs. This kind of setup is ideal for applications where a specific hardware configuration is needed or users only occasionally need the high compute capacity.

However, commercial cloud services charge, by the hour, for CPU time. In some settings, such as a large organization with many users, it might be more cost effective for the organization to purchase hardware to create its own private cloud. This is where open-source cloud frameworks such as Eucalyptus,[3] OpenNebula [5] and Nimbus [4] enter the picture. These software products are designed to allow an organization to set up a private group of machines as their own cloud. In this work, we analyze these three open-source cloud frameworks. We selected these three because they represent three different points of interest in the design space of this particular type of open-source cloud.

In this analysis, we will first briefly address the previous comparisons which have been made of these cloud architectures. These previous works largely focused on the feature set of these clouds. We will briefly summarize that work. In this analysis, however, we wish to go beyond feature comparisons. We will discuss how these

software frameworks act as managers that stand in the middle of a number of other software components. Understanding how these pieces fit together is critical for understanding any cloud framework. Next, we will analyze how core decisions in the basic architecture and overall structure of Eucalyptus, OpenNebula and Nimbus impact the kind of settings and applications for which each framework is most suitable. Third, we will identify several opportunities for improving these software stacks by identifying some of the challenges that are common to all three.

2 Previous Work

In looking for comparisons between Eucalyptus, OpenNebula and Nimbus, we found several papers which, in some manner, gave comparisons of some aspects of at least some of these systems. [17], [21], [12], [8], [23], [9] Most of this previous work focused on the feature set of these software projects. In Table 1, we summarize the basic feature comparisons from these prior works.

The open source projects developing Nimbus, OpenNebula and Eucalyptus are all rapidly expanding. Previous work as recent as 2009 does not include some of the new features developed by these projects. For example, the Nimbus project has recently added support for KVM hypervisors, which is not noted in some of the aforementioned citations. In such cases, we update the table to as recently as possible. However, we we fully expect any table of features, including ours, to become obsolete. This is especially the case with regard to support for standard API (such as EC2). Moreover, the standard API themselves often change, rendering prior support obsolete. Nevertheless, for features, Table 1 is the starting point of our comparison.

3 The Cloud Computing Stack

Before delving into the structures that make Eucalyptus, OpenNebula and Nimbus unique, we will make a quick overview of the entire cloud computing software stack. These three cloud infrastructures only make up part of the software stack needed to make a cloud computing system work. In many ways, software such as Eucalyptus, OpenNebula and Nimbus stand in the middle a large number of other components. Furthermore, there are often many options for these components. We include an overview of the software pieces in Figure 1.

In a generic open-source cloud computing system, we can identify six basic components. First, we have

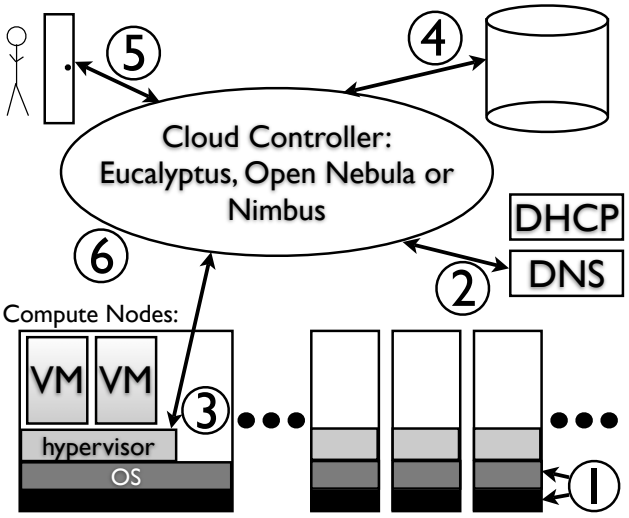

Figure 1. Abstract Cloud Architecture

The layout of the different parts of a generic cloud computing system. Note that the role of the cloud control software is to coordinate all of these pieces and to provide sufficient abstraction so that a user can simply request VMs with minimal concern for how these components must interact. The numbers in this diagram correspond to the components as we list them in this section.

hardware and operating systems that are on the various physical machines in the system. While these must be set up properly for any software system, we make especial note of the physical hardware and the base operating system for two reasons. First, if the processors of the physical nodes do not have the needed hardware extensions to run pure virtualization, this limits the system to paravirtualization only. While such a cloud system can be setup, it greatly limits both the speed of the VMs and the flexibility available in choosing software components for the system. Second, open-source frameworks, unlike commercial frameworks, must be flexible enough to work with many underlying systems. (Whereas, commercial clouds only need their system to work with the hardware that they have.)

The second component is the network. This includes the DNS, DHCP and the subnet organization of the physical machines. It also includes virtual bridging of the network that is required to give each VM a unique virtual MAC address. This bridging is accomplished using programs such as `bridge-utils`, `iptables` or `ebtables`. Moreover, in addition handling the physical nodes, DHCP and DNS processes must be configured, in concert with the cloud framework, to handle the MAC and IP addresses of virtual nodes as well.

Table 1. Cloud Architectures Compared

	Eucalyptus	**OpenNebula**	**Nimbus**
Disk Image Options	Options set by admin	In private cloud, most libvirt options left open.	Depends on configuration
Disk Image Storage	Walrus, which imitates Amazons S3	A shared file system, by default NFS, or SCP	Cumulus (recent update from GridFTP)
Hypervisors	Xen, KVM (VM Ware in non-open source)	Xen, KVM, VMware	Xen, KVM
Unique Features	User management web interface	VM migration supported	Nimbus context broker

The actual setup of these components depends heavily on whether one is using OpenNebula, Nimbus or Eucalyptus, as these systems can have different expectations for both physical and virtual network configuration.

The third component is the virtual machine hypervisor, (also known as a Virtual Machine Monitor or VMM). Popular VMMs include Xen and KVM, which are open-source, VirtualBox, which has an open-source version, and VMware, which is commercial. These programs provide a framework which allows VMs to run. In addition to the actual VMM itself, each of these cloud frameworks relies on a library called libvirt, which is designed to provide a facility for controlling the start and stop of VMs. However, the abstraction is not perfect, as each VMM has unique options that must be set. As such, the inputs to libvirt can differ slightly depending on the VMM (and VMM version used) used. For this and other reasons, the different cloud frameworks support different subsets of the hypervisors.

The fourth component is an archive of VM disk images. In order to usefully run VMs, a virtual hard drive must be available. In cases where one is simply creating a single VM on a single physical machine, a blank disk image is created and the VM installs an operating system and other software. However, in a cloud framework, where it is expected that hundreds of VMs will be constructed and torn down in a short amount of time, it is impractical to do a full OS install on each one. For this reason, each cloud system has a repository of disk images that can be copied and used as the basis for new virtual disks. In any given cloud, we must make a distinction between template disk images and runtime disk images. The template disk images are those stored in a disk image repository to be used for multiple VMs. When a VM is spawned, one of those templates copied and is packaged into a disk image appropriate for the given hypervisor. Usually, this involves adding a swap partition and padding the disk image to the appropriate size. It is the actual runtime image that is used by the virtual machine.

The fifth component is the front-end for users. There must be some interface for users to request virtual machines, specify their parameters, and obtain needed certificates and credentials in order to log into the created VMs. Some front-ends perform various types of scheduling by giving users an allotment of resources which they are not allowed to exceed. Other front-ends implement standard API such as EC2. We note that the front-end is one of the most customizable pieces of the entire system.

The last component is the cloud framework itself, where Eucalyptus, OpenNebula and Nimbus are placed. This framework processes inputs from the front-end, retrieves the needed disk images from the repository, signals a VMM to set up a VM and then signals DHCP and IP bridging programs to set up MAC and IP addresses for the VM.

4 Recurring Themes

Before proceeding with our comparison, we wish highlight four main ideas that consistently reoccur when looking at these programs. First, is the above idea of a complete cloud computing software stack. A cloud control system sits in the middle of a huge number of other components. Indeed, the actual cloud controller is only a small part of the overall system. Having such a high number of software interactions makes compatibility a constant issue. We are also lead, by this, to our second major theme, which is customizability. This idea is to be expected, given that these are open-source projects. Part of the appeal of setting up a private cloud, as opposed to using a commercial one, is that the administrator can have more control over the system. One of the most important questions to be asked about each of these frameworks is the degree to which customization is allowed for both administrators and users. Most notably, support for standard API interfaces is often one of these customizable components. For example, OpenNebula permits a front-end that uses a subset of the EC2 interface as an

option, but also lends itself to customized web front-ends, through its XML-RPC interface.

From this, we reach our third idea, which is the degree of transparency in the user interface. One of the hallmarks of the cloud idea in the commercial setting is the "black-box" nature of the system. The individual user, in a "pure cloud" is not aware of, or cares, where, physically, his VMs are running. In a more custom-made open-source setting, however, opportunities exist for a greater degree of explicit management with regard to the underlying configuration of physical machine and the location of VMs on them. The degree to which users can be permitted to examine and work on these underlying components varies among these systems and can often be fine tuned by administrators who customize the front-end.

Lastly, we note that open-source software frameworks, in many settings, allow for a much closer interaction between users and owners of computational resources. The users of a small to medium-sized company with its own cloud are far more likely to know and interact with the cloud administrator then if they were using a commercial cloud. This allows for users to be granted a certain level of trust that cannot be extended to commercial users. The level of trust allowed to users is a key distinction between clouds. We also note, however, that this is another example of a customizable component of some these systems.

5 Open Source Clouds

Earlier we quickly described some of the current features of Eucalyptus, OpenNebula and Nimbus, and more such features can be found by checking the documentation of these programs. We will now analyze the primary differences in the overarching structure and guiding philosophy of these frameworks. This is to achieve two things. First, the feature sets of these pieces of software change frequently. It is more interesting to examine the aspects of these projects that will not change as rapidly. Second, we wish to shed some light on the kinds of settings in which each of these frameworks is more suitable. (In cases in which a commercial version of the relevant cloud product exists, we are referring only to the open-source version.)

Eucalyptus

Eucalyptus is designed to be an open-source answer to the commercial EC2 cloud. Eucalyptus, for its front-end for users, includes a program called euca2ools, which is very similar to Amazon's EC2 front-end programs. However, some versions of EC2 itself, as well as Rightscale are also compatible front-ends. The overall design specification of the cloud have been published,

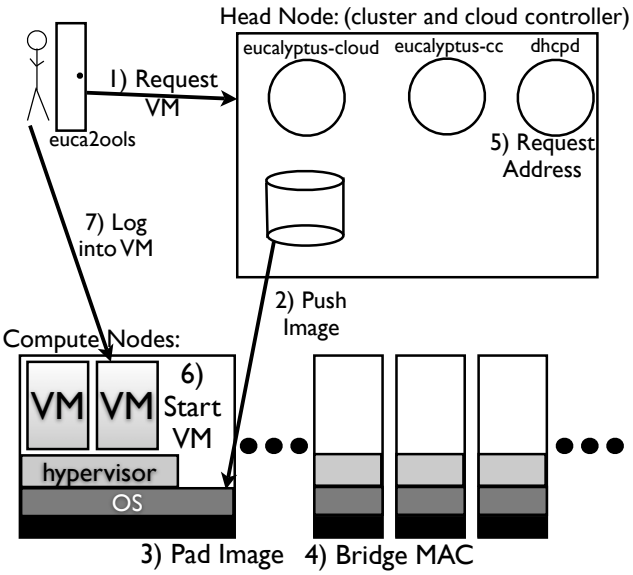

Figure 2. Eucalyptus

The steps for constructing a virtual machine in a configuration of Eucalyptus: 1) A user uses the euca2ools front-end to request a VM. 2) The VM template disk image is pushed to a compute node. 3) This disk image is padded to the correct size and packaged for use by the hypervisor on the compute node. 4) The compute node sets up network bridging to provide a virtual NIC with a virtual MAC. 5) On the head node the dhcp is set up with the MAC/IP pair. (Note: Technically this is STATIC mode. But other modes are similar.) 6) VM is spawned on the VMM. 7) The user can now SSH directly into the VM.

[20], along with an overview of the most recent version, [26] and much of the feature documentation is available online. [3] In figure 2, we give the steps taken in spawning a VM in a typical installation.

With regard to the overall philosophy underlying Eucalyptus, the system is very much designed for the corporate enterprise computing setting, a fact confirmed by the language used on its website and in its documentation. [20] All told, the structure of Eucalyptus confirms this focus. First, there is a very strong separation from user-space and admin-space. Root access is required for everything done by the administrator on the physical machines themselves. Users are only allowed to access the system via a web interface or some type of front-end tools. (The default is for this Eucalyptus' own euca2ools) With few exceptions, Eucalyptus attempts to protect users from as many of the complexities of the underlying systems as possible. For authentication of the euca2ools, users download a zip file with the needed keys and follow short instructions

on how to load them. Once included scripts set up certain environment variables, the euca2ools will work. Similarly, rather then expose the complexity of disk configurations available under libvirt, the administrator sets 5 configurations for available processors, memory and hard drive space, and the user must choose one of these sizes for each of their VM. In this way, users can be more easily protected from the complex underlying systems.

The software configuration also leans more toward decentralizing resources, insofar as possible. The system allows for multiple clusters, such that while there is a single head node for handling user interfaces, there can be multiple cluster controllers. This works particularly well if groups of machines in the cloud are physically separated from each other. Furthermore, Eucalyptus implements a distributed storage system called Walrus which is designed to imitate Amazon's S3 distributed storage. Essentially, users are assigned a cap for the amount of Walrus storage they are allowed to use. The storage is separated into structures which, if needed, can be distributed throughout the cloud. Each VM's running disk image, however, is stored locally on the compute node. This storage mode further increases the decentralization by allowing VMs, once started, to run independently of all other machines.

Eucalyptus also assumes that the administrators of the machines have some leeway to configure the network in accordance with the expectations of one of the network configurations of Eucalyptus. For example, if the administrator is using "SYSTEM", it is assumed that the external network is configured to accept new random MAC addresses and will assign IP addresses to them. Not all networks, especially secured ones, will necessarily do this. The other configurations assume that the cluster controller is allowed to operate its own DHCP server. Again, not all network setups will like this. Moreover, all these components must be configured with address ranges that are all in agreement with each other. As such, network configuration can present a real challenge in many network settings. Eucalyptus works best when each cluster is its own subnet, with its own reserved address range on the wider network.

The highly decentralized design of Eucalyptus, with multiple clusters, distributed storage, and locally stored running virtual disks, lends itself to a large number of machines. Second, as far as possible, the internal technical details are hidden from users, catering to persons whose primary focus might not be computer science. The web administration interface caters to settings where there is a large number of users to administer, including built in "signup" features in which users of the cloud might not know the administrator. Internal maintenance assumes undisputed root access to cloud machines. Optimal network configuration incorporates the ability to carefully control network address spaces, and most optimally, setup a dedicated subnet for Eucalyptus machines. All these features lend themselves to conditions which are more likely available in a corporate enterprise setting.

OpenNebula

OpenNebula tends to a greater level of centralization and customizability (especially for end-users). The steps for spawning a VM are shown in Figure 3. The configuration in this figure is the default configuration which is documented on their website. [5] However, there is a huge amount of customizability that is permitted in OpenNebula. Specifically, the idea of OpenNebula is a pure private cloud, in which users actually log into the head node to access cloud functions. This interface is a wrapper around an XML-RPC interface, which can also be used directly. We note that a front-end interface, such as EC2, can be appended to this default configuration.

The customization available in OpenNebula affects both users and administrators. From the administrator's perspective, the most striking customization available is in the shared file system used to store all of OpenNebula's files. OpenNebula, by default, uses a shared file system, typically NFS, for all disk images files and all files for actually running the OpenNebula functions. (We note that it can also use simple SCP to transfer files.) The advantage of this is that it exposes more of the underlying features of libvirt to the cloud users and administrators. This most notably includes things such as live VM migration. In practice, we also found that this centralization made the system easier to administer.

The customization of OpenNebula also is made available to users if one stays with the default idea of a private cloud. In order to spawn a VM, the user provides a configuration file containing parameters which would be fed into the VMM command line. This allows for memory, processor, network and disk resources to be requested for essentially any configuration. However, the downside to this kind of customizability is that it is easy for the user to make a mistake. This is especially the case if the underlying VMM configuration is in any way unusual or the network has special MAC or IP configuration requirements.

OpenNebula is also very centralized, especially in its default configuration with an NFS filesystem. The compute nodes do not need a large amount of hard disk resources. However, the NFS, can constitute a bottleneck for resources and requires a large amount of disk space. At the time of this writing, the OpenNebula

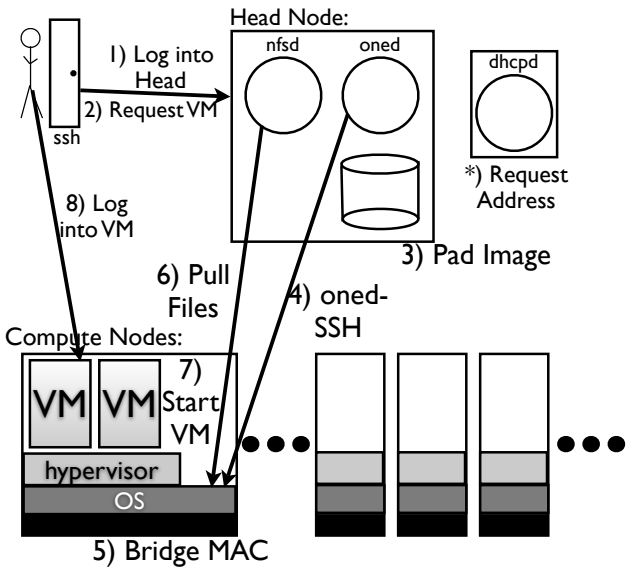

Figure 3. OpenNebula

*The steps for constructing a VM in a configuration of OpenNebula: 1) A user uses ssh to login to the head node. 2) The user uses the onevm command to request a VM. 3) The VM template disk image is copied and a copy is padded to the correct size and configuration within the NFS directory on the head node. 4) The oned process on the head node uses ssh to log into a compute node. 5) The compute node sets up network bridging to provide a virtual NIC with a virtual MAC. 6) Files needed by the VMM on the compute node will be pulled to the compute node via the NFS. 7) VM is spawned on the VMM. 8) The user can now SSH directly into the VM. *) Typically, the dhcp server is handled separately from the OpenNebula configuration.*

documentation recommended 1TB of disk space in the shared file system per 64 cores in the cloud. Of course, NFS can be made more efficient by devoting resources to the head node. However, this has a potential to get expensive, which might not be appropriate for settings relying on open-source software.

We point that customization can affect security. Of course, NFS does not encrypt any of its traffic. This can be a serious problem if the network is open to packet sniffing. Of particular note is the fact that OpenNebula relies on the administrator user of the cloud (default is called oneadmin) to be able to cleanly ssh into any compute node. To insure this, public and private keys are placed within the shared NFS directory. Of course, there are multiple network settings which can deal with this, such as using another shared or distributed file system, using the SCP option instead, tunneling the NFS connections or arranging a subnet that outside computers can't snoop. Nevertheless, the default arrangements more or less assume a fairly trusted setting, such as a private LAN.

All of these design ideas combine to form as system that (in its default configuration) is most conducive to a small to medium sized set of reasonably trusted users. Moreover, the level of customization available to administrators and users means that this kind of system works best when for users who know what they are doing from a technical perspective and can, therefore, take advantage of the added available features. Alternatively, the administrator can use an optional front-end like EC2 to protect the users. However, this still requires an administrator be able to customize the system for it. We also note that this level of customizability is suitable for researchers of computer science who wish to experiment with combining cloud systems with other technologies, such as SGE or Condor. Other available, non-standard, customizations include scheduling front-ends [25] [24] and sending overflow cloud requests to Amazon's EC2 [19].

Nimbus

The Nimbus project explicitly advertises itself as a "science" cloud solution. [4] [15] [14] Nimbus is also affiliated with the Globus Project and uses Globus credentials for the authentication of users. Until recently, Nimbus relied on GridFTP (a Globus project) to be used as a disk image repository. In their newest version, they are shifting to Cumulus, a system, like Eucalyptus' Walrus, compatible with S3. [30] Like OpenNebula, Nimbus is incredibly customizable. Figure 4 shows the steps in spawning a VM in a "typical" configuration.

The customization available in Nimbus, however, has a slightly different focus. One reasonable, albeit imperfect generalization, is that a large number of the customizations available in OpenNebula pertain directly to the underlying technical details of VM creation, whereas for Nimbus these details are more protected. OpenNebula basically allows for switching nearly every component, including the underlying file system, the DHCP, the front-end. Moreover, in the default configuration, much of the customization is available to users and administrators. Nimbus, on the other hand, leaves most of the customization to the administrator and not to the user and has several more components which are constants. These components include the image storage, (previously GridFTP and now Cumulus), the use of Globus credentials for all user authentication, and the requirement that the running Nimbus process can cleanly SSH into all compute nodes. Similarly, while Nimbus is very flexible in the number and types of virtual networks that can be set

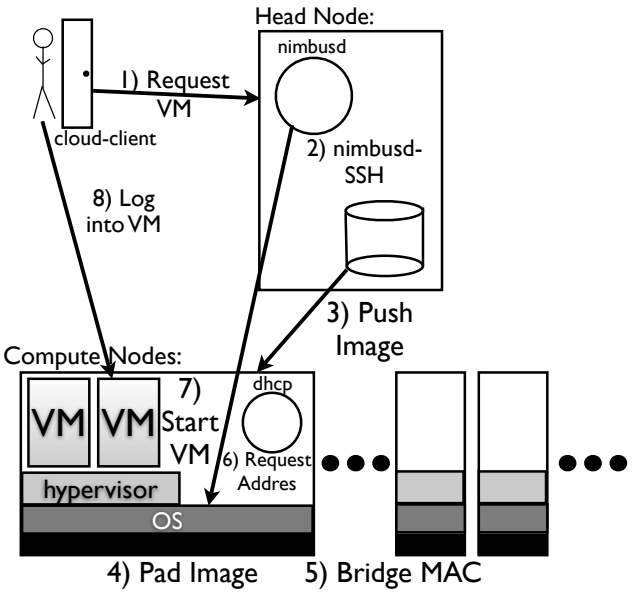

Figure 4. Nimbus

The steps for constructing a virtual machine in a configuration of Nimbus. 1) A user uses cloud-client to request a VM. 2) Nimbus will ssh into a compute node. 3) The VM template disk image is pushed to the compute node. (In the newest versions of Nimbus, this will be done using a distributed storage similar to S3 and Walrus.) 4) On the compute node, the disk image is padded to the correct size and configured. 5) The compute node sets up network bridging to provide a virtual NIC with a virtual MAC. 6) A dhcp server on the compute node is configured with a MAC/IP pair. 7) VM is spawned on the VMM. 8) The user can now SSH directly into the VM.

up [16], the underlying physical mechanism for doing so is much less flexible, (at least as of this writing), and involves a DHCP server on each compute node with Nimbus choosing a random MAC address.

Lastly, among these three pieces of software, Nimbus is the one which probably pays the most attention to capacity allocation and capacity overflow. The ability to give different users different lease limits as a means of scheduling comes standard with Nimbus, whereas it is an add-on for the other two. Second, the idea of allowing EC2 or another cloud the ability to pick up excess demand is heavily researched with Nimbus. [18] [10] This capacity is similar to previous research into "federated" clouds. [22]

Given all of these ideas, Nimbus sits somewhat in the middle of Eucalyptus and OpenNebula on the customization chain. There are a large number of options for user and administrators in deploying the cloud, but fewer of those options pertain to the nitty-gritty of the underlying software stack. The security level is slightly higher then in OpenNebula, due to the required integration of Globus certificate credentials. A facility is incorporated into the system for sharing capacity between clouds, if desired. However, Nimbus is not so open as OpenNebula, which exposes large amounts of the underlying software in the default "private cloud" configuration. Nimbus' level of openness is ideal for the scientific community, which would be most conducive to sharing cluster time, but most users probably do not want to contend with the oddities of the underlying software configuration.

Summary

Generally speaking, Eucalyptus is geared toward a private company that wants their own cloud for their own use and wants to protect themselves from user malice and mistakes. OpenNebula is geared toward persons interested in the cloud or VM technology as it own end. Such persons would want a VM sandbox so they can try new and interesting things on the computational side. OpenNebula is also ideal for anyone that wants to stand up just a few cloud machines quickly. Nimbus looks toward the more cooperative scientific community that might be less interested in the technical internals of the system, but has broad customization requirements. Such a community would be more likely to be familiar with the Globus Toolkit and would be more conducive to sharing excess cloud time. Table 2 summaries some of the general trends identified.

Of course, these are generalizations. However, we believe that they succinctly capture the overall philosophy of these projects, if one is aware of their dynamic nature, as well as the many options available. These factors make the situation a little more complex with regard to assessing performance, however. Therefore, the generalization above is for philosophy rather then performance of each of these projects.

6 Future Opportunities

In addition to analyzing the kind of settings in which each of these projects is most appropriate for cloud management, we wish to identify three opportunities that exist for all of these frameworks.

Scheduling Our first opportunity for research is in the area of VM scheduling. We note that OpenNebula and Eucalyptus, as of this writing, in their default configurations, do not do any real form of scheduling, in the sense of negotiating priority for processors. (To be precise, Eucalyptus does give each users a cap for space in the Walrus distributed storage.) Rather, if the re-

Table 2. Summary of Results

	Eucalyptus	OpenNebula	Nimbus
Philosophy	Mimic Amazon EC2	Private, highly customizable cloud	Cloud resources tailored to scientific researchers
Customizability	Some for admin, less for user	Basically everything	Many parts except for image storage and globus credentials
DHCP	On cluster controller	Variable	On individual compute node
Internal Security	Tight. Root required for many things.	Looser, but can be made more tight if needed.	Fairly tight, unless deploying a fully private cloud.
User Security	Users are given custom credentials via a web interface	User logs into head (unless optional front-end used)	Users x509 credential is registered with cloud
An Ideal Setting	Large group of machines for bunch of semi-trusted users	Smaller group of machines for highly trusted users	Deploy for less to semi-trusted users familiar with x509
Network Issues	dhcpd on cluster controller	Admin must set manually but has many options	dhcpd on every node and Nimbus assigns MAC

sources for a requested VM are available, it is allocated, if not, not. Nimbus allows for user to be given a cap on the number and size of VMs which they are allowed to create. Requests that exceed a particular user's limit are not honored. Of course, a simple front-end doing the same thing could easily be added to Eucalyptus and OpenNebula. In the case of OpenNebula, we note an active project that does this. [25]

This idea presents a interesting algorithmic problem for open-source clouds that is not present in commercial clouds. For a commercial cloud the variable of price can be used to simplify the idea of user priority. A user simply has the resources that they will pay for. If the demand for resources changes, then the price can be raised or lowered. For example, Amazon.com allows users to buy "spot instances" which allow users to specify a price they are willing to pay. When the demand reaches a low enough point, Amazon.com will start instances at that price. [1]

However, private clouds do not have the same type of price variable. Rather, there is a set of users that have, for some reason or another, access to the system, between which resources must be negotiated. Furthermore, this problem is further complicated by the fact that eviction is not an available option. This is in contrast to high-throughput engines such as Condor. [27] For Condor, the basic mechanism for resolving resource conflicts it to evict a job. For most grid engines, the mechanism is to delay the onset of a job. However, with these cloud systems, the expectation is that the VMs requested will be available for the duration of the scheduler's lease. A threat that the VM would disappear makes the whole system useless. Also, many applications (such as MPI jobs) require groups of VMs and it does little good to delay a part of them. As such, the mechanism for private cloud scheduling is the decision, yes or no, for a particular requested lease. This is an online algorithm problem, since the decision must be made upon the request, without knowledge of future requests. Furthermore, preemption is usually not an option, because a VM lease, once granted, is assumed granted for the duration of the lease.

This sets up an intriguing algorithmic problem. What online algorithm can schedule the requests for VM resources, in absence of money, such that computational resources remain available for high priority users, but resources are also not left idle unnecessarily? This becomes more of an issue the more that users are expected to be given access to the private cloud. Furthermore, the classic methods of eviction and preemption, used for grids and Condor, do not apply in the same way.

Networking When networking is discussed in the context of private clouds, it is usually referring to the virtual networking of the VM. Nimbus, in particular, has great interest in "One-click clusters," or groups of VMs, spawned all at once, complete with a virtual network between them. We briefly note the similarity to previous work on "virtual clusters" [28]. However, an important contrast is that the virtual clusters are designed to run on grids. As such, the physical resources given to the virtual cluster are not constant and can be "pulled" into use more dynamically. A "one-click cluster" in a cloud system is a static group of VMs. This underlies the concept of clouds providing leases to VMs, rather then job scheduling. There is another aspect of networking, however, the interaction between the virtual and physical networks upon which the cloud is run. While all three of these cloud systems do a reasonably good job of providing users with the option of creating interesting and useful virtual network configurations, the assumptions made regarding the physical network configuration, often outside the cloud administrator's control, varied greatly. While, as mentioned

briefly before, the best solution is to give the cloud administrator full control of the network space where the cloud exists, (for example, by providing a separate subnet) this is not always practical. In practice, we found that the most frustrating aspect, in every case, of setting up these programs was configuring the cloud controller to cooperate with network parameters that were not entirely under our control.

Briefly, each of the clouds handles assigning IP addresses to VMs slightly differently. Eucalyptus either relies totally on the external network to assign IPs to any random MAC address ("SYSTEM" mode) or handles the assigning of IP addresses through a DHCP server that is on the cluster controller node. OpenNebula relies on the cloud administrator to set up their own DHCP server that will know what to do with the MAC address ranges for which OpenNebula is configured. Nimbus has each compute node run a DHCP server on that compute node for all VMs on that node. This DHCP server is configured by Nimbus such that the desired IP is assigned to the matching MAC address just randomly produced.

It is easy to see the conflicts that can ensue if the network administrator and the cloud administrator are not the same person. In Eucalyptus, if the routers are configured to filter traffic that is not to certain IP addresses, the router tables must be coordinated with Eucalyptus' IP/MAC lists. In OpenNebula, the cloud administrator must be allowed to set up their own DHCP server and know what IP and MAC address combinations will not conflict with the rest of the network. For Nimbus, there is the same filtering problem as in Eucalyptus, as well as issues that can arise if MAC address filtering is used. (Nimbus only creates random MAC addresses.) Also, these problems are for the default cloud configurations. The more the cloud customizes, the more one can expect conflicts.

As such, all of these cloud frameworks have the potential problem of network conflicts if they are not the only thing in their own subnets. There are many probable solutions to this. First and most simple is, in the case of enterprise level deployments, only use designated cloud subnets. A second idea is to make the cloud controller programs more flexible, not only with regard to available virtual networks, but also with regard to potential interactions with an externally administered physical network. In any case, making this problem easier would remove one of the major obstacles to widening the community of users for each of these products.

Leaky Abstractions As we stated before, one of the main considerations of these three cloud frameworks is the degree to which they separate the user from the nitty-gritty of the underlying software implementation. However, the case of the VMM is one key way in which this abstraction is not perfectly maintained. Specifically, for each of these cloud frameworks, it is necessary to know which VMM (And, in some settings, which version of the VMM) is being used in order to properly configure a VM instance. Even in Eucalyptus, the sample disk images supplied by Eucalyptus contain alternate kernels for Xen or KVM.

Additionally, problems can exist because VMM configuration details can vary between underlying operating systems. Libvirt, while serving as a suitable common interface for constructing and monitoring VM, does not abstract out these configuration variations between VMM types. As such, users must be made aware of some aspects of the underlying configuration in order to properly configure VMs. This is more of the problem in OpenNebula then in Eucalyptus, since (again, under the default configuration) OpenNebula requires users to supply more of the details of the VM. In Eucalyptus, many, but not all, of these details must be set for all users by the administrator.

Practically speaking, these issues can be separated into two aspects. First, is the difficulty of constructing disk images. However, this is not a tremendous issue, given that, basically, the user must know to include a xen-enabled kernel in their configuration if they are using xen. Also, the disk image construction difficulty extends beyond the underlying VMM and includes much more interesting issues related to producing a software stack that is both useful and of reasonable size. There is plenty of prior research on this though, one highly successful attempt to resolve these issues is the CernVM. [6] The second difficulty is, in practice, more centered on the abstraction issue. Simply put, when underlying details of the system affect the configuration of higher-up components, it makes it more likely the user or the cloud administrator will make mistakes. It is this difficulty that presents a challenge of finding a better abstraction for the libvirt layer, so that commands to the hypervisor can truly be the same format from cloud setup to cloud setup. A start to such a solution might include making the libvirt layer more intelligent with regard to translating generic VM instructions into hypervisor specific instructions. This would, however, require separate libvirt implementations for each hypervisor/OS combination. However, such a layer would make it easier to configure systems with heterogeneous hardware/hypervisor setups.

7 Conclusion

In analyzing these various open-source cloud computing frameworks, we find that there are salient philosophical differences between them regarding the overall scheme of their design. While the huge amount of customization possible, as well as the ongoing feature development of these tools, makes it difficult to make accurate absolute statements about them, we have, however, identified certain strong tendencies in the focus of each of these products. Moreover, we have pointed out several opportunities for improving these frameworks.

8 Acknowledgements

We wish to express our gratitude to Dr. Paul Brenner and Mr. Stephen Bogol who assisted us in testing these software frameworks. We also are grateful to all persons involved in mailing lists, development and documentation of the Eucalyptus, OpenNebula and Nimbus projects.

References

[1] Amazon EC2 Spot Instances. http://aws.amazon.com/ec2/spot-instances/.

[2] Amazon elastic compute cloud (EC2). http://aws.amazon.com/ec2/.

[3] Eucalyptus Home Page. http://www.eucalyptus.com/.

[4] Nimbus Home Page. http://www.nimbusproject.org/.

[5] Open Nebula Home Page. http://www.opennebula.org/.

[6] P. Buncic, C. A. Sanchez, J. Blomer, L. Franco, S. Klemer, and P. Mato. CernVM - a virtual appliance for LHC applications. *Proceedings of Science - XII Advanced Computing and Analysis Techniques in Physics Research*, November 2008.

[7] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brand. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616, June 2009.

[8] D. Cerbelaud, S. Garg, and J. Huylebroeck. Opening The Clouds: Qualitative Overview of the State-of-the-art Open Source VM-based Cloud Management Platforms. *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, (22), 2009.

[9] P. T. Endo, G. E. Gonçalves, J. Kelner, and D. Sadok. A Survey on Open-source Cloud Computing Solutions. *Brazilian Symposium on Computer Networks and Distributed Systems*, May 2010.

[10] G. V. M. Evoy, B. Schulze, and E. L. Garcia. Performance and Deployment Evaluation of a Parallel Application in an on-premises Cloud Environment. *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, 2009.

[11] W. Gentzsch. Sun Grid Engine: Towards Creating a Compute Power Grid. *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, 2001.

[12] P. C. Heckel. Hybrid Clouds: Comparing Cloud Toolkits. 2010.

[13] D. Hilley. Cloud Computing: A Taxonomy of Platform and Infrastructure-level Offerings. Technical Report 13, Center for Experimental Research in Computer Systems - Georgia Institute of Technology, April 2009.

[14] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the Use of Cloud Computing for Scientific Workflows. *SWBES 2008*, December 2008.

[15] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications, August 2008.

[16] K. Keahey and T. Freeman. Contextualization: Providing One-Click Virtual Clusters. *eScience 2008*, December 2008.

[17] Z. Lei, B. Zhang, W. Zhang, Q. Li, X. Zhang, and J. Peng. Comparison of Several Cloud Computing Platforms. *Second International Symposium on Information Science and Engineering*, pages 23–27, 2009.

[18] P. Marshall, K. Keahey, and T. Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010)*, May 2010.

[19] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Elastic Management of Cluster-based Services in the Cloud. *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, June 2009.

[20] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-source Cloud-computing System. *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009.

[21] B. P. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. *Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51, 2009.

[22] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Systems Journal*, 2008.

[23] B. Sotomayor, I. Foster, R. S. Montero, and I. M. Llorente. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 2009.

[24] B. Sotomayor, K. Keahey, and I. Foster. Combining Batch Execution and Leasing Using Virtual Machines. *HPDC 2008*, June 2008.

[25] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Capacity Leasing in Cloud Systems using the OpenNebula Engine. *Workshop on Cloud Computing and its Applications 2008 (CCA08)*, October 2008.

[26] T. Tan and C. Kiddle. An Assessment of Eucalyptus Version 1.4. Technical Report 2009-928-07, Grid Research Centre, University of Calgary, Canada, 2009.

[27] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In *Grid Computing: making the global infrastructure a reality*, pages 299–350. John Wiley and Sons, 2003.

[28] E. Walker, J. P. Gardner, V. Litvin, and E. L. Turner. Personal adaptive clusters as containers for scientific jobs. *Cluster Computing*, 10(3):339 – 350, September 2007.

[29] L. Wang, G. V. Laszewski, M. Kunze, and J. Tao. Cloud Computing: a Perspective Study. *Proceedings of the Grid Computing Environments (GCE) workshop*, November 2008.

[30] L. Wang, J. Tao, M. Kunze, D. Rattu, and A. C. Castellanos. The Cumulus Project: Build a Scientific Cloud for a Data Center. *In proceedings of International Conference of Cloud Computing and Applications*, October 2008.