GENI in the Cloud

by

Marco Yuen
B.Sc., University of Victoria, 2006

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

GENI in the Cloud

by

Marco Yuen
B.Sc., University of Victoria, 2006

Supervisory Committee

ABSTRACT

Computer networking researchers often have access to a few different network testbeds (Section 1.2) for their experiments. However, those testbeds are limited in resources; contentions for resources are prominent in those testbeds especially when conference deadline is looming. Moreover, services running on those testbeds are subject to seasonal and daily traffic spikes from users all round the world. Hence, demand for resources at the testbeds are high. Some researchers can use other testbeds in conjunction with the ones they are using. Even though each of the testbeds may have different infrastructures, and characteristics, in the end, what the researchers receive in return is a set of computing resources, either virtual machines or physical machines. Essentially, those testbeds are providing a similar service, but researchers have to manage the credentials for accessing the testbeds manually, and they have to manually request resources from different testbeds in order to setup experiments that span across different testbeds.

This thesis presents GENICloud, a project that enables the federation of testbeds with clouds. Computing and storage resources can be provisioned to researchers and services running on existing testbeds dynamically from an Eucalyptus cloud. As a part of the GENICloud project, the user proxy (Section 3.4) provides a less arduous method for testbeds administrators to federate with other testbeds; the same service also manages researchers credentials, so they do not have to acquire resources from each testbed individually. The user proxy provides a single interface for researchers to interact with different testbeds and clouds and manage their experiments. Furthermore, GENICloud demonstrates that there are, in fact, quite a few architectural similarities between different testbeds and even clouds.

Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to express my most sincerest gratitude to my supervisors, Dr. Yvonne Coady and Dr. Rick McGeer, for their support, and supervision. Special thanks to Dr. Yvonne Coady for her patience and believing in me, and to Dr. Rick McGeer for presenting me with the amazing opportunity to work on this project and providing me with resources I need. A special thank-you to Dr. Andy Bavier from Princeton University for his technical advice and assistance.

Finally, I would like to thank all my friends and family for their support, patience and getting me away from the computer.

DEDICATION

To my parents

Chapter 1

Introduction

During the development of GENICloud, I setup a development environment on Emulab[91], a networking testbed. The development environment includes a cloud, and the cloud only consists of two nodes. Emulab has a policy to swap out idle nodes, as well as, nodes that have run for a period time, typically ten hours, despite the fact that the node may still be active. Swapping out a node is the same as releasing the nodes back to the resource pool so others can use. One day, I was trying to swap in the nodes for my tiny private cloud and continue the development, but there weren't any nodes free on Emulab. All of the nodes were being used, and the development of GENICloud was halted for two days until there were nodes freed up.

The scenario described above is no stranger to a lot of researchers. Testbeds like Emulab, and PlanetLab are essential to networking researchers' works. However, testbeds are often limited in resources, and contentions for resources are high especially prior to deadlines of conferences or experiments gone haywire. Testbeds and resource providers (e.g., data centers) are constantly investing more resources into their networks. However, the growth does not catch up with the users demands; also, there is the often unpredictable spike in traffic. The network traffic spike can be caused by different factors. One is the seasonal factor. Couple weeks or even months prior to major holidays (e.g., Christmas, Chinese New Year), users can put a lot of strain on different network services, because they need to get last minute shopping done or talk to their relatives. In fact, traffic spike can even come in different daily cycles. Companies can have network services that serve many different customers from different time zones. Customers from one zone could be winding down just when customers from other time zone about to start their work days. Companies need some way to manage their resources efficiently to reduce cost and, at the same time, satisfy their customers. Not only just the networking researchers demand for more computing resources, researchers from other scientific disciplines often require massive amount of computing resources. Some prime examples are physicists, astronomers, and biologists. Physicists, especially particle physicists, have been building computing infrastructures to support their experiments for years. In the recent years, the completion of Large Hadron Collider[53] at CERN[2] provides a great platform for particle physicists ways to conduct high-energy physics, and there are different experiments[27, 29, 45, 33] planned to run at the collider, those experiments can easily generate Petra-bytes of data. As such, the physicists are constantly in demand for computing resources to process those data. Astronomers who are in the field of observational astronomy often acquire data of celestial objects through different instruments (e.g., radio telescope, infrared telescope). Similar to the physics experiments, the observational data collected by astronomers are enormous, and the astronomers need computing resources to store and process those data. Projects like SETI@Home[66] aims to provide astronomers computing resources by having other users to donate the processing power of their home computers.

1.1 Cloud Computing

Cloud computing is a model of computing in which a shared pool of computing resources are provisioned to users on-demand with minimal management effort or intervention from service provider. The shared pool of computing resources in a cloud can be compute nodes, storage, applications, services and etc. Resources in a cloud are usually hosted on a network managed by a third party service provider instead of the users' local network, and the users interact with those resources via the Internet. Perhaps one of the most recognized use of cloud computing is its scaling capabilities. Cloud users can dynamically allocate more resources from the cloud to their applications if their applications require more resources. Those extra resources can be released back to the cloud any time when the spike period is over. Cloud computing abstracts away the infrastructure details of computing resources. To the users, they do not have to invest in, configure and maintain any computing infrastructure which is quite costly; they can just use those resources. Under such service model, cloud computing provides the computing infrastructure as a service (IaaS). Cloud computing also supports other service models such platform as a service (PaaS) and software as a service. Under the PaaS service model, a cloud can provide a development as

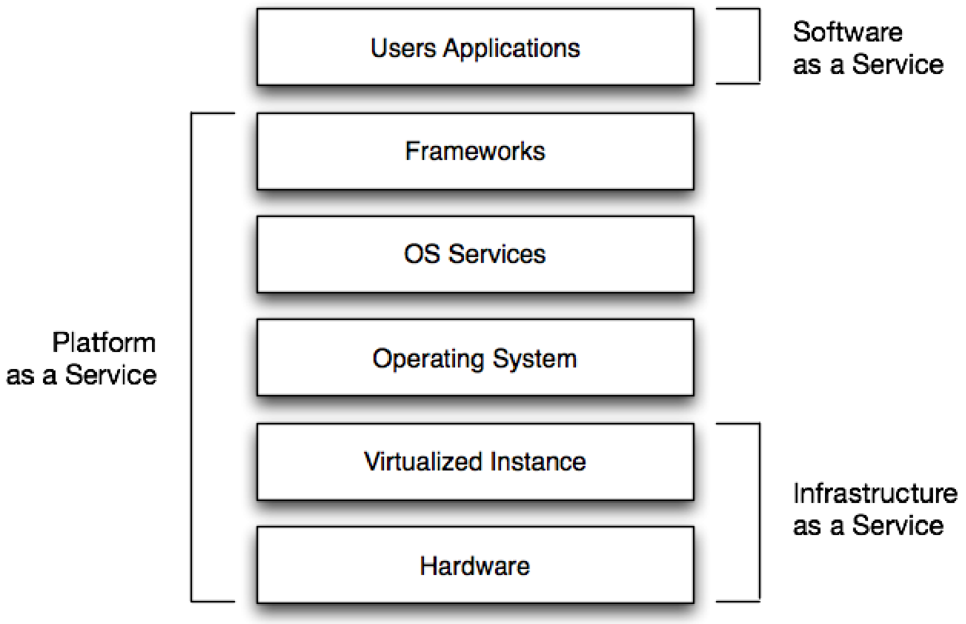

Figure 1.1: Cloud Computing Services

well as a deployment platform for the users' applications or services. For SaaS, users can subscribe to software already deployed on the cloud. In Figure 1.1, a generalized view of the cloud computing stack is displayed alongside with the different service models. The service models provide users varying degree of freedom. Users are free to choose the model that will suit their need the best. Networking Testbeds can benefit from cloud computing because extra resources can be provisioned on-demand by the cloud to, say, PlanetLab with minimal to no cost, given the clouds' administrators agree to allow PlanetLab's users to use the resources.

1.1.1 Technologies

There are several key technologies that make the cloud computing model possible. Perhaps, one of the most important technologies that cloud platforms employ is hardware virtualization[86]. Essentially, hardware virtualization allows individuals to create virtual machines, and the virtual machines created are "*efficient, isolated duplicate* of the real machines"[79]. A virtual machine provides an isolated environment for programs to run and the environment it provides is nearly identical to the host machine. Also, the programs running within the virtual machine should only see minimal performance degradation. Cloud computing relies on different virtualization technologies like Xen[35], and KVM[12]. By creating virtual machines on top of physical machines, a cloud computing platform can more efficiently and dynamically provision computing resources to users. Unlike traditional cluster environments where users get the whole physical machine even though they probably will not utilize all of its resources, virtualization allows multiple virtual machines to be run on a single machine. Moreover, virtualization enables greater flexibility in terms of virtual machine customizations; users can customize the specifications (e.g., cores, memory, disk space) and even the operating systems of the virtual machines to tailor fit their needs. Since users have the freedom to create and customize a number of virtual machines, the cloud computing platforms also provides a management interface to help users to manage the life cycles of those virtual machines. Life cycles being the instantiation, and destruction of the virtual machines. Cloud platforms usually expose the management interface using either RESTful or SOAP-based web services[55, 30] or both. The users, through the web services, can control each individual virtual machine that they instantiate, and they can query the status of those virtual machines. With all the virtual machines instantiated, the users can access those instances through various network protocols (e.g., SSH[92], RDP, RFB). The most prominent protocol is probably SSH. The SSH protocol allows users to establish a secure shell session with the virtual machines. Within the shell session the users have complete control over the virtual machines; the users can install software, deploy their services, or develop their application.

1.1.2 Commercial Clouds

There are many different commercial offerings of cloud computing. Rackspace[19], RightScale[21], FlexiScale[5] are just a few of the commercially available cloud vendors. However, the most successful of them all would be Amazon Elastic Compute Cloud(EC2)[1]. Amazon EC2 exposes its cloud platform through a web service. Users can use that web service to acquire computing resources, and as EC2 being one of the commercial clouds, it also defines a pricing model. The users are charged by the hour per instance, and depending on the specification of the instances different rate is charged. So more powerful instances will cost more per hour. In addition to instance charges, Amazon EC2 charges the users for data transfer. Both upload and download bandwidth are metered and the users will be charged according to their usage.

Accompanying the cloud services, Amazon EC2 offers other services built on top of its cloud platform. For example, Amazon offers a monitoring service called CloudWatch to users, so they can monitor the performance, resource utilization and usage patterns of their EC2 instances. Combining with the CloudWatch service, the user can enable the auto scaling capability for their services running on the cloud. When CloudWatch and auto scaling are enabled, the users can define conditions, so EC2 can scale up during heavy demand and scale down when the demand subsides. Other services that Amazon provides to their EC2 users include Amazon Elastic Block Store (EBS), which provides a persistent storage service for EC2 instances, and Elastic IP, which sets up an IP address such that it is associated with the user's account but not a particular instance. All of these extra services may have additional charges, but most of these services are not available in open source cloud platforms, one of them is discussed next.

1.1.3 Introduction to Eucalyptus

Eucalyptus is an open-source software framework that enables the offering of computer infrastructure as a service (IaaS) to other users. Specifically, different compute clusters can be grouped together and offer their compute resources to users as a cloud. As a result, those resources that are offered can be accessed in a more convenient and unified way. Eucalyptus' main goal, however, is to allow research community to explore different research questions regarding cloud computing, as the commercial alternatives (e.g., Amazon EC2, Rackspace) do not open up their infrastructures for research purposes. Amongst Eucalyptus' features are the dynamic provisioning of compute resources, namely, virtual machines, storage services similar to that of Amazon Simple Storage Service (S3) and Amazon Elastic Block Store (EBS) which allow users to persist data within the cloud and access those data with practically from anywhere, and Eucalyptus' interfaces are modelled after one of the most successful commercial clouds—Amazon Elastic Compute Cloud (EC2), as a result, Eucalyptus is compatible with tools that are specially written for Amazon EC2. However, the underlying implementation of Eucalyptus is very different from Amazon EC2.

Eucalyptus is made up of different components, cloud controller, storage controller, cluster controller, node controller, and Walrus, an Amazon S3 compatible storage service. The overall architecture of Eucalyptus is shown in Figure 1.2. As the figure describes, only the cloud controller is the only component that interacts with the users; users will not interact with other controllers. However, the users can

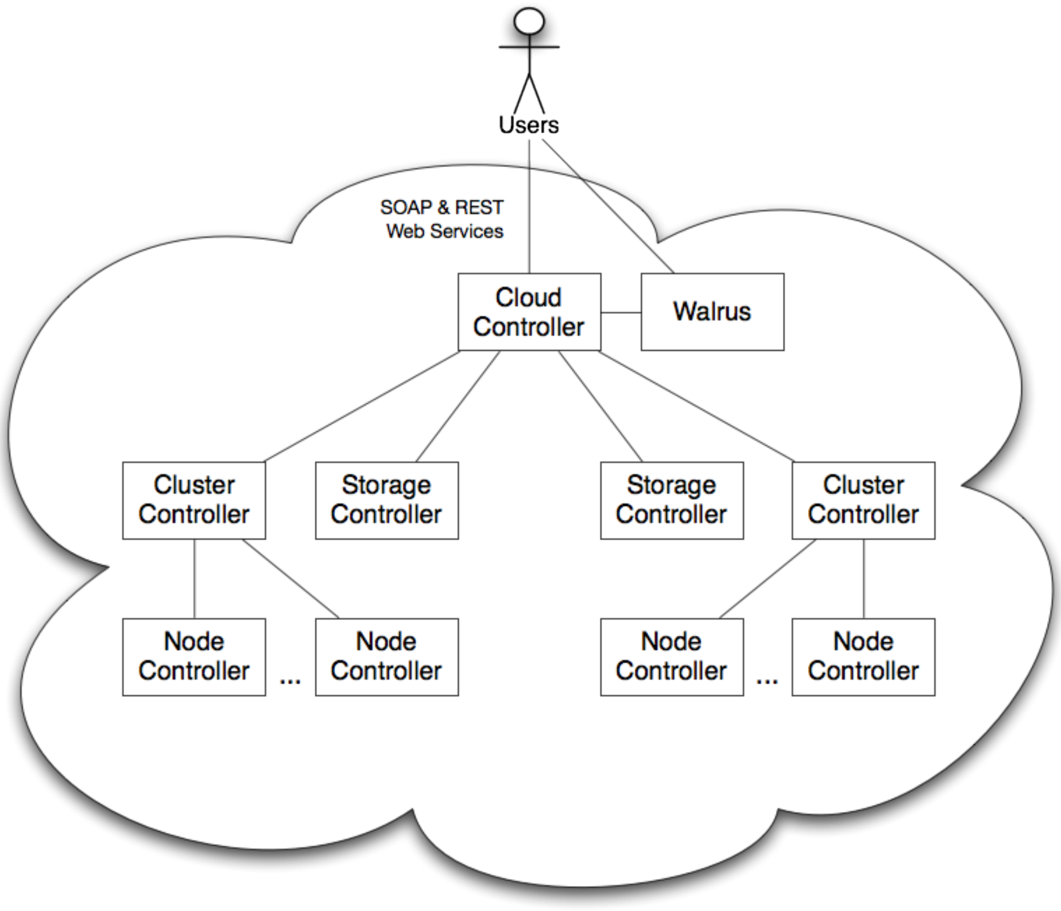

Figure 1.2: Components in Eucalyptus

communicate the Walrus service directly for storing and retrieving data. The cloud controller implements an interface that is compatible with Amazon EC2. In fact, Amazon EC2's command line tool as well as software libraries for EC2 can be used to control Eucalyptus clouds. The cloud controller handles all cloud related operations; for example, users use the cloud controller to create new instances, upload disk images to the cloud, and etc. Cluster controller has the same responsibilities of the head node of a cluster. The cluster controller keeps track of the states of its nodes and make sure they are alive, responsive and ready to serve. The cluster controller also performs very rudimentary scheduling; it will decide which nodes, under its supervision, will be used to run the virtual machines requested by the users through the cloud controller. In the recent stable release of Eucalyptus (1.6.x), the cloud controller can support multiple clusters. Therefore, multiple clusters can be aggregated and form a cloud, and shielded users from underlying details of the computing infrastructures. The node controller is run on every node in a cluster or clusters. The node controller will wait for cluster controller's commands, and will do the grunt work of instantiating virtual machines or shutdown/restart/remove virtual machines when instructed to. The storage controller is running on each cluster; its function is to provide a block-storage service to the nodes similar to that of Amazon Elastic Block Store. For example, volumes can be allocated using the storage controller and those volumes can be accessed as if they are block devices, so the users can mount those volumes inside their virtual machines. The volumes will not be destroyed when the virtual machines are terminated. As such new virtual machines can be brought up and access those volumes by mounting them; the data within the volumes can then be reused. The last component is Walrus. It is a storage service similar to Amazon Simple Storage Service. Walrus uses a buckets analogy where data are stored into buckets. Users are free to create and remove buckets. Once the buckets are created, the users can put and delete any files they want from those buckets. In Eucalyptus, Walrus is where all the disk and kernel images are stored. The Walrus service can be accessed through different separate command line tool, without going through the cloud controller.

Eucalyptus provides different ways to allow users to access its services. The cloud controller provides a SOAP-based and RESTful web services for users to access the resources in an Eucalyptus cloud. Eucalyptus also distributes command-line tools for users to access and manipulate those resources. A simple usage scenario is that the users can use the command-line tools to discover the resources, for example, the specifications of the virtual machines (e.g., number of cores, memory available, etc.), and the images that are available for the virtual machines. After that, the users can use the command-line tools to create virtual machines instances. The same scenario can be done using either the SOAP-based or RESTful web services as well. As mentioned before the users have access to the Walrus storage service; Walrus also allows access through SOAP-based and RESTful web services and command-line tools.

1.2 Network Testbeds

For networking researchers, there are multiple networking testbeds available for them to conduct experiments and deploy their services. Even though the testbeds may have different underlying architecture, network topology, or extra features, those testbeds

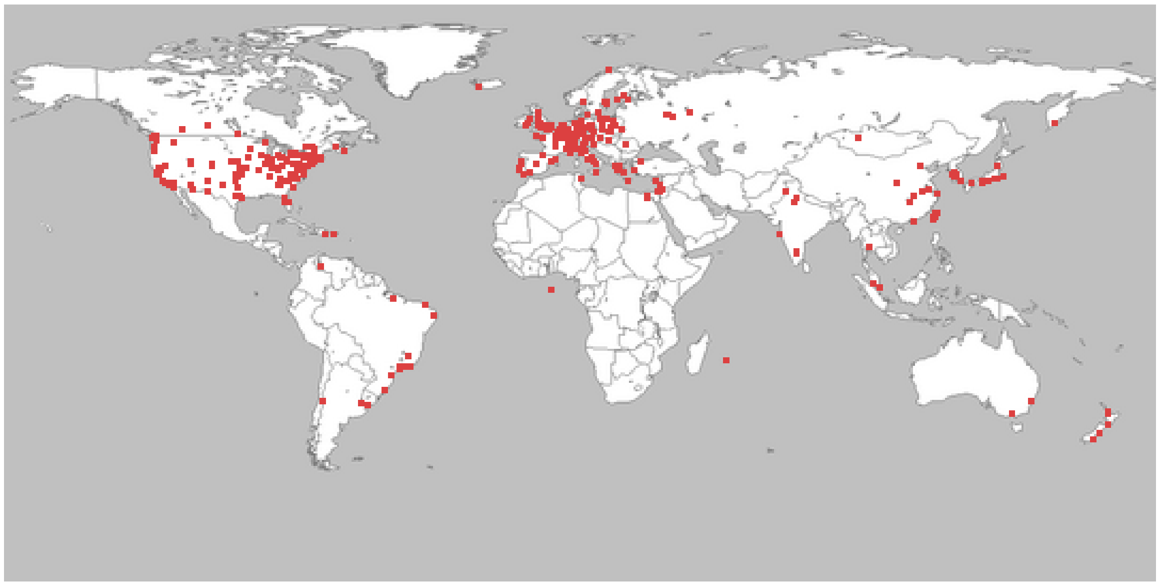

Figure 1.3: PlanetLab Sites

all, in essence, provide the same service to researchers—provisioning of computing resources. In the end, what the researchers get are computing resources, and most of time those are all they need—similar to cloud computing, and for those experiments where extra features or control are needed, the researchers can choose the testbed that fits their requirements. For the rest of this section, a few of the testbeds along with their characteristics are illustrated.

1.2.1 PlanetLab

Testing distributed applications and network services in a global scale have always been difficult, because deploying such applications and services could have adverse effects on the Internet. Service providers are reluctant to open up their infrastructure to researchers for the fear of interfering with existing services to their clients[32]. As such, researchers have to resort to testing their distributed applications and services in a smaller scale or in a simulated environment. But the environment does not allow the researchers to fully realize the real potential of their applications or how resilient their services are under a real network environment. To provide a more realistic platform for researchers, PlanetLab is testbed for exploring disruptive technologies in a global scale[76, 37]. The testbed has a total of 1085 nodes at 498 sites. The sites are geographically distributed around the globe. Figure 1.3 shows a world map with the sites represented as red dots. Researchers can deploy their distributed applications and services on PlanetLab in a more realistic environment in that the nodes are geographically distributed, accessible from the Internet, and expose to actual Internet traffic and conditions.

PlanetLab is built as an overlay network on top of the Internet. Overlay networks have been used to solve various problems. For example, Distributed hash table like Chord[85] is organized as an overlay network for `<key, value>` storage; Resilient Overlay Network[31] is an application-layer overlay network to improve fault tolerance in distributed applications; M-Bone[52], an overlay network for facilitating IP multicast, and many other. An overlay network creates a virtual topology on top of the physical network. Overlays have the flexibility to be re-programmed to accommodate new features or capabilities (e.g., a new routing algorithm) without having to go through the ISP and asking for resources and permission. Each node in the PlanetLab is multiplexed out through the use of virtualization. Multiple virtual machines can be running on the same PlanetLab node and those virtual machines can be belonged to different experiments or services. The virtualization technology that PlanetLab utilizes is Linux-VServer[14]. Linux-VServer creates an isolated jail environment for individual virtual machine within a node, and in PlanetLab parlance such environment is called a sliver (Section 1.2.1). Since PlanetLab expects the services running on it are long running services instead of one-time globally scheduled services, those services, as a result, will get a fraction of each node's resources (e.g., CPU cycles, memory); unlike the traditional paradigm in cluster/grid computing where the services running will get all of the resources of the node. PlanetLab is managed centrally by PlanetLab Central (PLC) located at Princeton, New Jersey. PLC has control over all of the nodes in different sites; PLC also maintains a database of network traffic within PlanetLab. The network traffic database can be used to provide an audit trail in case of Acceptable Use Policy violations. If one of the experiments on PlanetLab has violated the PlanetLab Acceptable Use Policy (AUP), staff at PLC can terminate the experiments remotely and the violators will be held responsible for their actions.

Slices and Slivers

As discussed before, services running on PlanetLab receive a fraction of the node's resources. Therefore, whenever the users want to acquire computing resources from PlanetLab, what they get is a set of distributed virtual machines—distributed virtualization [37]. PlanetLab defines a concept of treating the set of distributed virtual machines as a single, compound entity called a slice. The concept comes from the fact that whenever a service is running on PlanetLab, it receives a *slice*(virtual machines running on different nodes) of the PlanetLab overlay network. Individual virtual machine within a slice is called a sliver. While PlanetLab's concept of slices centers around a set of distributed virtual machines, one can generalize the concept to encompass other types of resources within the slices. With GENICloud, we have expanded the concept of slices to include Eucalyptus virtual machines, and, in the future, storage capability. So a slice in GENICloud can have both PlanetLab resources and virtual machines from an Eucalyptus cloud. The users can log into individual sliver in a GENICloud slice and conduct their experiments.

1.2.2 German Lab (G-Lab)

In Germany, the researchers are facing similar problems when it comes to networking research and innovations. They feel like the current Internet does not keep up with the ever changing network technologies, and that the Internet is not a suitable platform for deploying and testing innovative experiments and services. In order to provide researchers the right facility for their future Internet technologies, G-Lab[87] is created to fill in the missing piece. G-Lab provides a German-wide facility for research in architectures, protocols, and services. As of this writing, the G-Lab facility has 174 nodes at six different sites. G-Lab has an architecture similar to that of Eucalyptus. At the top level, there is the central node (Eucalyptus cloud controller); it is responsible for resources provisioning and scheduling and boot images management. At each site of G-Lab, there is a head node (Eucalyptus cluster controller). The head node manages the local nodes and executes commands from the central node. At the node level, the G-Lab users have the options to either use whole physical node, thus, granting access to physical hardware, or create virtual machines on the node, similar to Eucalyptus.

1.2.3 Open Resource Control Architecture (ORCA)

ORCA[48, 47] is a control framework for the Breakable Experimental Network(BEN) [34]. BEN relies on dark fibers as its underlying architecture, and it is a metro-scale testbed that interconnects three different universities (UNC-CH, Duke and NCSU). The reason why BEN is breakable is because it is using dark fibers, so it will not interfere with production traffic, and the researchers will have the freedom to deploy their disruptive technologies on BEN without worries. ORCA, being a control framework for BEN, is responsible for acquiring resources for users. In fact, its vision is to be viewed as an "Internet operating system"[48]. ORCA provides a leasing abstraction to users, thereby allowing them to lease computing resources from different providers (e.g., PlanetLab, Grid, or virtual machines). A lease can be seen as an agreement between the resource provider and the lease holder, and it will grant the lease holder an exclusive control to a set of resources and the lease is renewable.

1.2.4 CoreLab

CoreLab[73] is a PlanetLab-based network testbed located in Japan. Since it is based on PlanetLab, CoreLab has a lot of similar features like slice abstractions, slice management framework, isolation of resources through virtualization, and etc. With that being said, CoreLab strives to improve upon the flexibility problem in PlanetLab. The flexibility problem in PlanetLab stems from the fact that slivers are container based. While container based virtual machines offer higher performance and scalability[83], they lack flexibility in that the virtual machines share the kernel, network stacks and other resources with the host machines; the sharing of those crucial resources makes introducing changes almost impossible. To cope with the inflexibility in container based virtualization, CoreLab employs virtual machine monitor(hypervisor), specifically, Kernel-based virtual machine[12] as the virtualization technology. Hypervisor-based virtualization offers more flexibility while maintaining the isolation between virtual machines and between the host and its virtual machines. Moreover, the performance of hypervisor-based virtualization has improved tremendously over the years[71].

1.2.5 Emulab

Emulab[91] is another networking testbed similar to PlanetLab. However, its architecture is quite different from PlanetLab. Rather than having the testbed consists of geographically distributed nodes at different sites, Emulab testbed mainly consists of a cluster resided at the University of Utah. The control framework of Emulab have greater control of the network routing condition, since it is not an overlay network like PlanetLab. As such, Emulab users can define links between nodes, say, a VLAN or direct links between nodes. Users allocate nodes and links by submitting a NS file

```
set ns [new Simulator]
source tb_compat.tcl

set frontend [$ns node]
tb-set-node-os $frontend PCloud-euca-frontend
tb-set-hardware $frontend pc3000

set node1 [$ns node]
tb-set-node-os $node1 EucalyptusNode
tb-set-hardware $node1 pc3000

set link0 [$ns duplex-link $frontend $node1 100000kb 0ms DropTail]

$ns rtproto Static
$ns run
```

Listing 1.1: An example NS file

via the Emulab's web user interface. Within the NS file, the users can specify the what kind of nodes, what kind of OS images should be loaded on those nodes, and what kind of links should exist between the nodes. Listing 1.1 shows an example of a NS file we use during the development of GENICloud. The NS file is what we used to deploy a small two-node Eucalyptus cloud on Emulab. In the NS file, we specified each node with different disk images, and create a direct link between the Eucalyptus cloud frontend and a Eucalyptus node. The disk images are custom made in order to reduce deployment overhead.

1.2.6 VINI

VINI[38] is a network testbed in the same vein as PlanetLab. But, it is more ambitious than PlanetLab in that VINI strives to provide researchers a even more realistic environment for researchers to evaluation new protocols, routing algorithms, services, and network architectures. In terms of realism, VINI can allow researchers to run existing routing software, so they can evaluate their services or protocols under realistic network environment. Another characteristic of VINI is the ability to expose experiments to realistic network conditions; often times, when deployed, the new experimental protocols and services will be in a shared environment where external network events will be generated by other protocols and services that exist in the same shared environment. Researchers will need to make sure their experimental services can handle those events that are considered outside influences and that those external influences will not affect the operations of the researchers' services. Asides from events that are introduced by third party, the experimental service should react properly to expected networking events such as link failure, or flash traffic. VINI allows research to inject network events to the network, so researchers can test their protocols and services without waiting for the events, say, link failure to happen. Another crucial characteristic of a testing environment is to allow experiments to be deployed and used by real users and hosts easily. That way, researchers can better understand their services and get direct feedback from the users which is invaluable for testing experiment protocols and services, but difficult to achieve with existing testbeds.

1.2.7 Great Plains Environment for Network Innovation

The Great Plains Environment for Network Innovation (GpENI)[84] is another networking testbed located around a regional optical network in the Midwest United States. GpENI has a very distinguished characteristic in that all seven layers of the OSI-model[93] are programmable. In other words, the whole GpENI testbed is programmable and can be completely customized to the researchers' needs. In fact, the way that GpENI achieves such flexibility is by cleverly leveraging other existing technologies. For example, GpENI uses VINI and programmable routers to provide programmable topologies; PlanetLab's SFA to provide layer 4 and 7 programmability; GENI User Shell (Gush)[9] is used as the frontend for users to control their experiments and facilitate resource discovery.

1.2.8 OneLab

OneLab[15] is an umbrella project from Europe. Its goal is to create a testbed for network research for the future Internet. OneLab encompasses many different testbeds in Europe or testbeds from different continents, one of OneLab's flagship testbeds is PlanetLab Europe. In a nutshell, PlanetLab Europe is the European version of PlanetLab Central as described in Section 1.2.1. PlanetLab Europe is not managed by PlanetLab Central which is located Princeton, New Jersey; instead it is managed by OneLab in Paris, France. One of OneLab's main objectives is to federate the different testbeds that are under its supervision. PlanetLab Europe has successfully federated with PlanetLab Central. Other testbeds that are in that federation include

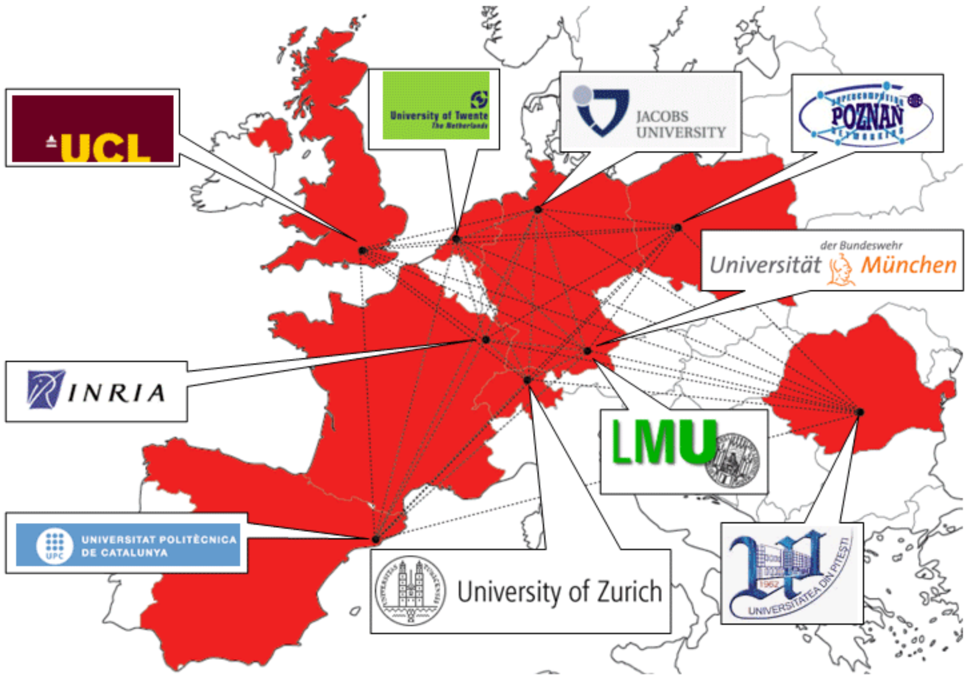

Figure 1.4: Sites in EmanicsLab

PlanetLab Japan[16] and EmanicsLab[4]. PlanetLab Japan is similar to PlanetLab Europe in that its governing body is located in Tokyo, Japan. Before the federation, it is a separate entity from other PlanetLabs around the world, especially PlanetLab Central. EmanicsLab is another one of the testbeds that is federated with PlanetLab Europe. EmanicsLab is based on `MyPLC`[62], which is a portable installation of PlanetLab Central. In other words, one can install the entire PlanetLab Central software stack on a single machine. As illustrated in Figure 1.5, the database, web server, boot server and API server are installed on a single host. Such configuration makes `MyPLC` extremely easy to deploy in different institutions including ones that are not part of the PlanetLab Consortium. Private institutions can easily setup their own private PlanetLab in house without exposing its private resources to other public PlanetLab(PLC) users just like they would if the institutions have joined the PlanetLab Consortium.

EmanicsLab is not quite as big as other testbeds(Figure 1.4); it has twenty nodes at ten different sites across Europe.

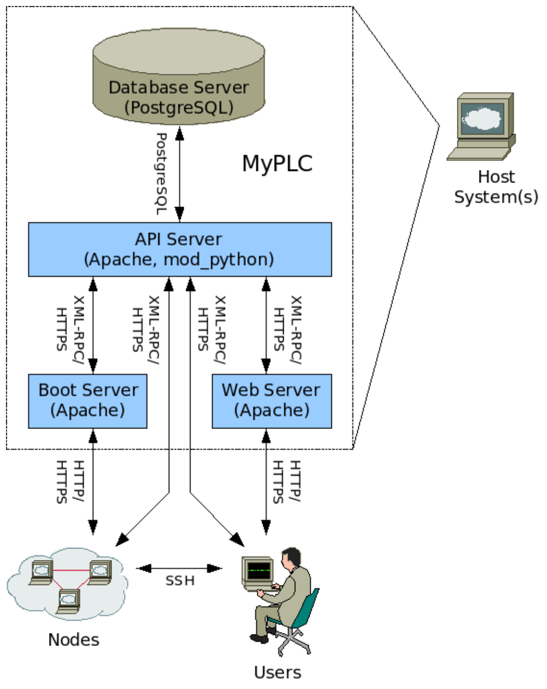

Figure 1.5: `MyPLC` Architecture

1.3 Introduction to GENI

GENI[6] stands for Global Environment for Networking Innovations, it is a project involving many institutions and researchers in the U.S. Its main goal is to build a network environment beyond the capabilities of the current Internet so that researchers can conduct disruptive experiments at-scale. Unlike existing testbeds which are mostly built as an overlay network on the Internet, GENI is designed from the ground up including its infrastructure. Resources on the GENI are not limited to only computer nodes; they can be mobile sensors, and wireless sensors. GENI provides a highly configurable and heterogeneous environment for researchers. GENI focuses its research from network architecture, network backbone to control frameworks. In fact, PlanetLab and Emulab are two of the control frameworks being selected as the control framework for GENI.

1.4 Motivation

GENICloud's goal is to allow the federation of heterogeneous resources like those provided by Eucalyptus[74], an open-source software framework for cloud computing, with GENI. When an Eucalyptus cloud is federated with GENI, all of its resources, including computing resources and storage resources, are readily available to GENI users. Under the federation of Eucalyptus and GENI, a more comprehensive platform is available to users; for example, development, computation, data generation can be done within the cloud, and deployment of the applications and services can be done on the overlay (e.g., PlanetLab). By taking advantage of cloud computing, GENI's users not only can dynamically scale their services on GENI depending on the demand of their services, they also can benefit from other services and uses of the cloud. Take a service that analyzes traffic data as an example; the service can deploy traffic collectors to collect Internet traffic data on PlanetLab, since it has many nodes deployed around the world. The traffic collected can be stored in the cloud. When the service needs to analyze the collected data, it can acquire computing resources from the cloud. Moreover, if coordination is required between the collectors and analyzers, a messaging bus[82] can be deployed on the cloud to facilitate communications between the two.

PlanetLab, being a part of the GENI project as one of the control frameworks, has high global penetration. While PlanetLab itself makes a great deployment platform, it has a few drawbacks. It lacks a sufficient data storage service. Some services and experiments require a huge amount of data or they need to persist a large amount of instrumental data; also, it lacks the computation power for CPU intensive services or experiments. GENICloud fills in the gap by federating heterogeneous resources, in this case, a cloud platform with PlanetLab.

Chapter 2

Related Work

Federating networked resources is not always thought of as a first class problem back when commodity computers are still not powerful enough to run more than a word processor and a terminal, and network bandwidth between those computers are still in snail pace compared to today's standard. However, as commodity are computers getting more and more capable and network speed is a lot faster and more affordable, privately owned clusters begin to spring up everywhere, since it is possible and affordable for individuals who do not have big corporate financial backing to deploy their own miniature clusters. There are even distributions like the Rocks Cluster Distribution[22], a customized Linux distribution, for building clusters easily using a set of commodity computers. And, since recent years, cloud computing has garnered a lot of interest from various communities, software solution like Eucalyptus is getting very popular to a point that Ubuntu, one of the most popular Linux distributions, released a specialized distribution for deploying a private cloud using Eucalyptus. The Ubuntu specialized distribution is called Ubuntu Enterprise Cloud[26].

Small to mid-size clusters start to spring and users of those clusters start to run into a similar that I ran into as mentioned in the introduction (Chapter 1)—the users need more resources. The clusters administrators have a few options; they can buy more hardware; they can refer users to other larger size clusters, or they can combine other small to mid-size clusters together—federation. Amongst the three options, federation is the most economical because the cost of new hardware can be pretty high and larger clusters may incur high service charges. This chapter covers various approaches and attempts to federate networked resources. It outlines a few concrete examples of existing testbeds federation and, at the end, other, more general, federation approaches.

2.1 Emulab and PlanetLab Federation

The Emulab's control framework has a PlanetLab portal. It has the ability to create Emulab experiments using PlanetLab slices. The portal takes care of slice provisioning for the users. Underneath the hood, the Emulab PlanetLab portal uses the PlanetLab API (PLCAPI) to create slices. The PlanetLab portal's goal is to allow Emulab users to deploy experiments on PlanetLab. However, there is no federation involved. The portal only goes one way—allowing Emulab users to use PlanetLab resources. PlanetLab users have no access to any of Emulab's resources. On PlanetLab's point of view, it has no idea that the users are originating from Emulab. Also Emulab users cannot create experiments that consist of resources from both Emulab and PlanetLab. If the users have decided to use the PlanetLab portal for an experiment, they can only use resources from PlanetLab.

2.2 VINI and PlanetLab Federation

The first prototype implementation of VINI is called `PL-VINI`[38], which is an implementation of VINI on PlanetLab. Recently VINI is federated with PlanetLab using the Slice-based Facility Architecture (SFA) which will be explained in Chapter 3. In other words, resources at VINI are available to users of PlanetLab. The approach that VINI employs to federate with PlanetLab is the same approach that GENICloud employs to make Eucalyptus resources available to PlanetLab users(describe in Section 4.1). Unlike Emulab's approach, implementing a PlanetLab portal, VINI's approach is much more integrated in that the communication can go both ways. This is made possible by Slice-based Facility Architecture (Section 3.2). With two-way communication, VINI users have access to PlanetLab's resources and vice versa. PlanetLab will be aware of the users are coming from VINI, hence, be able to establish a chain of responsibility.

2.3 OneLab and PlanetLab Federation

Federating all of the mentioned testbeds (PLE, PLJ, EmanicsLab) within OneLab is definitely a non-trivial engineering problem, but the problem is alleviated by the fact all of the testbeds are based on PlanetLab's code base. They all run PlanetLab software stack, and have the same APIs by default without any modifications.

However, such luxury does not exist for the GENICloud project. Even though one of GENICloud requirements is that the federating networks must implement the Slice-based Facility Architecture interfaces, so the federating networks can communicate with each other under common interfaces. Most of the time those networks being federated only need an extra software layer to make them SFA compatible. One prime example is Eucalyptus, which, by default, does not have any notion of slices. Within Eucalyptus all it knows about are individual virtual machines but not a collection of them. Hence, we have implemented a software layer, aggregate manager, that makes resources on an Eucalyptus cloud “slicable”.

2.4 Other Federation Approaches

Grid computing[57] can be thought of as an extension to computer clusters. In grid computing, clusters of computers are geographically distributed in different sites, and those sites are managed by separate administrative domains. However, federating those clusters entails solving many different complex problems[80] such as security, resource discovery, policy and etc. To deal with the federation problem, one of the solutions that come up is GridShib[89], which is the integration of Shibboleth[25] and Globus Toolkit’s[56] Grid Security Infrastructure (GSI)[90]. Shibboleth is a service that provides identify federation across multiple security domains. In other words, it provides single sign-on and uses a attribute-based authorization model to make access decision on the resources. Globus is the *de facto* software framework for Grid computing; it provides high-level services for computational grids. By using GridShib, Globus can make access decisions by querying the Shibboleth service for a given identity. When a user wants to access a grid through Globus, the user presents her X.509 certificate. Then, GridShib will contact the Shibboleth attribute authority, and retrieve a sets of attributes about that certificate. The access decision can then be determined by the attributes returned. GridShib has been proven to work with some existing grids, albeit it has some limitations[36, 75].

Another area where federating is getting more important is between business domains. As businesses are moving their software architecture to a more service oriented architecture[39], interactions between business services that span across multiple domains are becoming more prominent and inevitable. Service-Oriented Architecture (SOA) is set of design principles and paradigms that reduce the complexity of business integrations including service integration, data integration, enterprise informa-

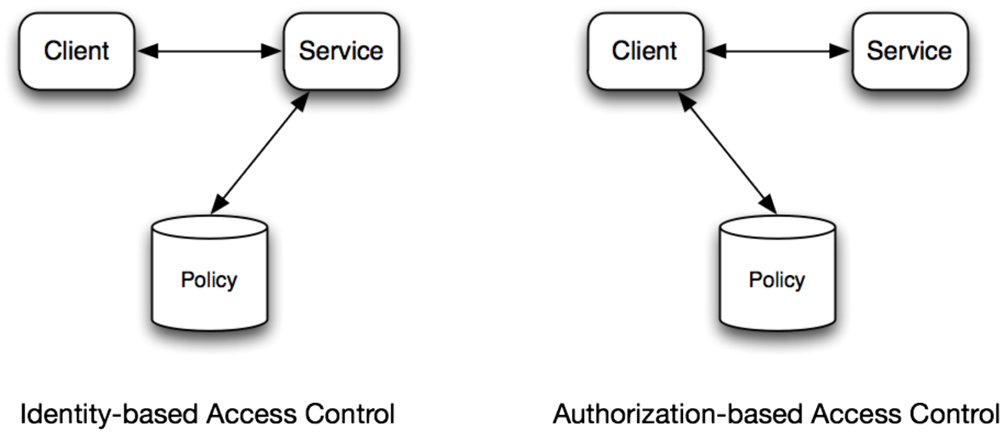
Client
Service
Policy
Identity-based Access Control
Client
Service
Policy
Authorization-based Access Control

Figure 2.1: Identity-based and Authorization-based Access Control

tion integration[70] and etc. One of the main principles of SOA is to have loosely coupled services where each service implements a functionality in the system, and the system consists of composing different services together. For example, one service is responsible for serving web contents; another service is responsible for authenticating users. Therefore, by composing those two services together one can get a web site with authentication capability. It is important to note that those services are standalone services—they contain no calls to other services. Such loosely coupled architecture relies on several technologies. One of which is Web services[30] and the set of corresponding specifications (WS-* family). One of those web services specification pertaining to federation is WS-Federation[59]. The specification defines mechanisms, and architecture for identify federation. Those mechanisms are used to broker information about identities, attributes, authentication and authorization.

The aforementioned federation technologies are all revolved around the idea of identities—they all try to federate identities from different security domains. [68, 64] point out that the federated identify approach contributed to the decline in scalability, security and increase in manageability for SOA. Instead, [64, 65] propose an authorization based access control as illustrated in Figure 2.1. In authorization-based access control, the users first talk to a policy engine, a service that can authenticate the users and determine their privileges, the users will receive a set of authorizations from policy engine, then the users can make requests to the service. While making requests to to a service, the users provide the authorizations along with the requests, then the service only needs to make sure those authorizations are not forged. On the other hand, in identity-based access control, the users will make request to the service first and, in turn, the service will contact the policy engine. Authorization-based access control does not have the inherent problems of identity-based access control mostly because each security domains only managing their users[64]. A reference implementation of a service, called Zebra Copy, that employs authorization-based access control[69] is available.

2.5 Summary

While federation is not a new problem, but it has been on the rise recently. The shift in computing paradigm from centralized massive data centers to more flexible distributed systems and constant demands for more computing resources from users raise the need for a more efficient way to federate distributed systems (e.g., testbeds). This chapter presents a few existing federated testbeds, as well as, other federation techniques. GENICloud has a slightly different approach to federation that aims to make federation simple.

Chapter 3

Design of GENICloud

GENICloud consists of multiple components. Some of those components involve varying degrees of modifications to existing codebase while some involve designing from the ground up. Nonetheless, throughout the development process of GENICloud various design decisions are made to make sure GENICloud is secure, efficient, and user friendly. However, since nothing is perfect, various design decisions also lead to some compromises. GENICloud utilizes many different technologies, and the components within GENICloud leverage different frameworks, libraries and architectures in order to speed up development.

This chapter will start off with the architecture upon which some of the GENICloud components are built. Then, different models of federation that are supported by architecture are explained. The explanation will use the PlanetLab implementation of the architecture as an example. A different model of federation that GENICloud uses is presented with design decisions and trade offs. The remaining of the chapter will discuss the different components that make Eucalyptus, the cloud computing framework, fits into the federation picture.

3.1 PlanetLab and Eucalyptus Comparison

Before discussing the design of GENICloud, a comparison between PlanetLab and Eucalyptus architectures will provide some insights into some of the similarities between the two seemingly disparate systems. The underlying infrastructures for both systems are quite different; PlanetLab comprises nodes scattered around the globe, and Eucalyptus consists of clusters. The architectures for both systems, however, bear some striking resemblances. Both PlanetLab and Eucalyptus at their cores start out with some computing resources, namely, physical machines that they can provision to users. However, instead of provisioning the physical machines to users, they both make heavy use of virtualization technology and provide users with virtual machines. PlanetLab uses Linux-VServer and Eucalyptus supports multiple technologies like Xen, and KVM. Virtualization allows services to share physical resources and allows the services to run for a long period of time. The virtual machines provisioned by both systems can be regarded as both development platform and deploy platform for services, although, PlanetLab discourages users to do any development on the PlanetLab nodes. Asides from the use of virtualization, the management method and interfaces are quite similar in both systems. PlanetLab and Eucalyptus provide their own remote API for users to easily acquire and manage their computing resources. For example, PlanetLab mainly employs XMLRPC[72] as the interface to PlanetLab, whereas Eucalyptus exposes its management interface by providing SOAP-based and RESTful web services. The core functionality of the remote API for both systems are pretty much the same; both interfaces allow users to discover the resources available in the systems; they both allow the users to obtain computing resources from the systems, and they both allow the users to manage and query their computing resources. Besides the remote API, PlanetLab and Eucalyptus both have their respective web GUI. Arguably, though, the PlanetLab web GUI is more powerful then the one in Eucalyptus, because web GUI in Eucalyptus only has the capability to show the users some of the resources available (e.g., disk images), but does not allow users to create and manage virtual machines. In both systems, the users access their computing resources through the use of public and private keys. Once logged into the computing resources from either PlanetLab or Eucalyptus, the users have root access to the resources, so the work flow that the users use on a system can be equally applicable to the other.

There is one profound conceptual difference between PlanetLab and Eucalyptus, and that is the lack of slices in Eucalyptus. As described in Section 1.2.1, PlanetLab has the concept of combining a set of virtual machines into a single, compound entity—a slice. However, Eucalyptus provides no such concept; it just treats each virtual machine as a separate entity even though some of the virtual machines may be related (e.g., part of a larger service). The concept of slices enable a more intuitive management of related virtual machines on the network. A service (e.g., content distribution network) can span across multiple virtual machines, and managing those virtual machines as a single entity is definitely more superior than managing those virtual machines individually.

3.2 Slice-based Facility Architecture (SFA)

SFA[78] is an architecture that defines a set of interfaces and data types which enable slice-based networks (e.g., PlanetLab, Emulab, etc.) to federate and be interoperable. Asides from the interfaces and data types, SFA also defines a few abstractions to aid developers in implementing the interfaces; one of the main abstractions is the idea of slices (Section 1.2.1). There are three interfaces that SFA defines, and they are registry interface, slice interface, and component management interface.

Registry Interface In a nutshell, a registry keeps track of all objects (e.g., users, slices, nodes, etc.) in a slice-based network. The registry interface abstracts away the implementation details of a registry, so it can be an actual database, or it can be a file, or anything the site developers see fit as a persistent layer.

Slice Interface This is an interface for manipulating slices. It provides ways to instantiate slices, to provision slices with resources, to control slices, and to query slices. This interface only controls slices as a whole, but not the individual slivers within the slices. Controlling of the slivers is done using the component management interface.

Component Management Interface A component, as defined by SFA[78], encapsulates a collection resources. Those resources can be physical resources (e.g., CPU), logical resources (e.g., sockets), and synthetic resources (e.g. network routes). This management interface manipulates the life cycle and states of a component such as, restarting the component or changing state of the component.

These interfaces are designed to be as generic as possible in order to encompass different slice-based networks.

GENICloud leverages this architecture by implementing a subset of the SFA interfaces for Eucalyptus, so GENICloud enables existing Eucalyptus clouds to federate with other slice-based networks or networks that implement the SFA interfaces. By implementing the SFA interfaces on top of Eucalyptus, GENICloud provides the abstraction of slices to Eucalyptus. One benefit of this federation is that users of a slice-based network can make use of resources at an Eucalyptus cloud. The current implementation of GENICloud works with PlanetLab. As of this writing, PlanetLab is the only slice-based network that implements the SFA interfaces. Details about the GENICloud implementation of the SFA interfaces are described in Section 4.1.

3.3 PlanetLab's SFA Implementation

Since SFA only defines a set of interfaces and data types but not implementation, developers are free to implement those interfaces however they want. As a result, any networks' administrators can choose to implement those interfaces and thereby enabling their networks to interoperate with other slice-based networks. There is a prototype implementation of SFA that is supported by PlanetLab.

PlanetLab's SFA implementation can be broken up into various managers and a registry that perform specific tasks as illustrated in Figure 3.1. Those managers that make PlanetLab operational are the aggregate manager, the slice manger and a registry server. As for all the PlanetLab nodes, each of them has a component manager running on top. The registry server, as briefly described in Section 3.2, maintains database of objects in PlanetLab, also it implements the registry interface defined by SFA. For the slice manager and aggregate manager, they both implement the slice interface defined by SFA. The component managers that run on all of the PlanetLab's nodes implement the component manager interface defined by SFA.

The managers perform different tasks. Even though aggregate manager and slice manager both implement the same interface, the slice interface, they interact with different components within the PlanetLab architecture. For the aggregate manager, it interacts with all the component managers, or in PlanetLab's case, all the PlanetLab's nodes. The aggregate manager understands the underlying details about managing and interacting with the actual computing resources and aggregates them so the computing resources can be managed by the slice interface. The slice manager, on the other hand, interacts with the users and aggregate managers. Whenever the users want to perform any operations related to slices, they talk to the slice manager which, in turn, talks to the aggregate managers, and since the aggregate managers implement the slice interface, the interaction between the slice manager and aggregate managers is simplified. It is important to note that there can be multiple aggregate managers running and the slice manager can talk to each of them individually (Fig-

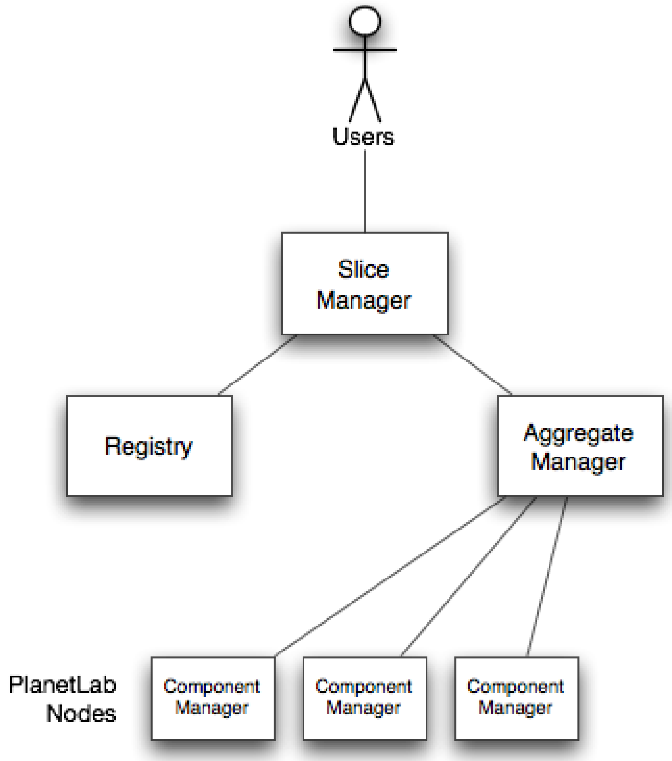

Figure 3.1: Managers in PlanetLab

ure 3.3). The slice manager can be seen as an aggregate of aggregate managers. The component managers running on all the PlanetLab's nodes enable individual nodes to be powered cycle and queried about their states. For example, one can find out whether a node is running properly, hence in the `boot` state, or in the `debug` state meaning the node encountered errors.

In order to create, provision slices or perform any operations with SFA, the users, currently, will have to use the Slice-based Facility Interface(`SFI`) tools[77]. SFI is a command line frontend to PlanetLab's implementation of SFA. Users can use it to discover resources in PlanetLab, create new slices, provision slices, and etc. Since SFI is the only frontend available for interacting with SFA, users of GENICloud will have to use it as well to interact with an Eucalyptus cloud when it is federated with PlanetLab. In the later section of this chapter, usage of SFI will be explained.

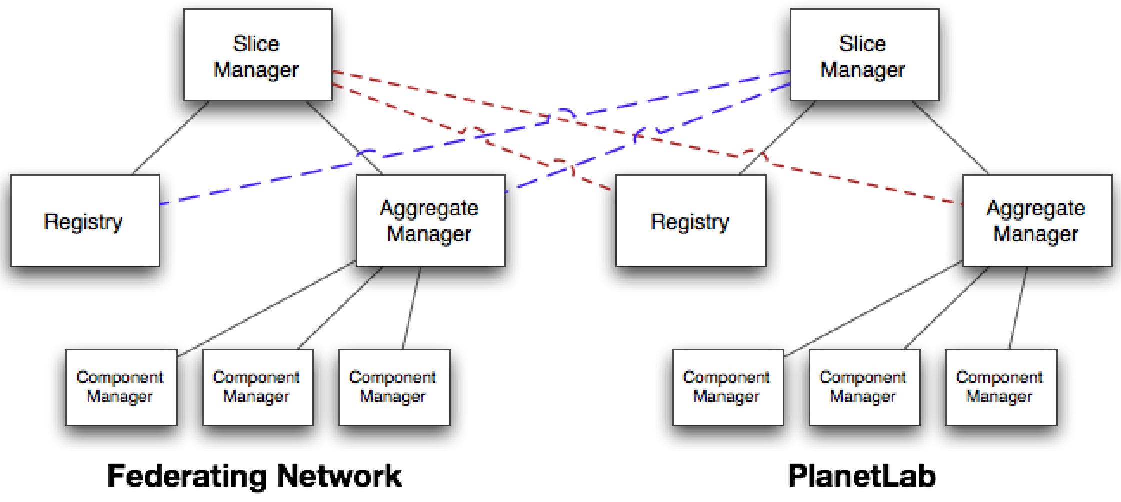

Figure 3.2: Complete Federation with PlanetLab

3.3.1 Federating with PlanetLab

After describing SFA, its interfaces, and its implementation in PlanetLab, this section will describe how a slice-based network can be federated with PlanetLab using SFA. Using current implementation of SFA supported by PlanetLab, a slice-based network can federate with PlanetLab in two ways. The first way is to implement an aggregate manager for the slice-based network to be federated. The second way is to implement all of the managers, namely slice manager, aggregate manager, and component manager, and a registry for the federating slice-based network. The second method essentially fuses both the federating network and PlanetLab together.

If a slice-based network chooses to federate with PlanetLab using the second method mentioned above, not only would its resources be available to PlanetLab's users, but also the users of the federating slice-based network will have access to PlanetLab's resources. This is because the slice manager in PlanetLab would have had access to the aggregate manager of the federating slice-based network and vice versa as illustrated in Figure 3.2. In addition, in Figure 3.2, you can see that the registries on both PlanetLab and the federating slice-based network are accessible to each other. Therefore, each site can access each other objects, for example, user accounts, credentials, other services, and etc. This approach is more suitable for the federating slice-based networks to share similar properties as PlanetLab, because the federating networks will have access to PlanetLab's internals. As such any incompat-

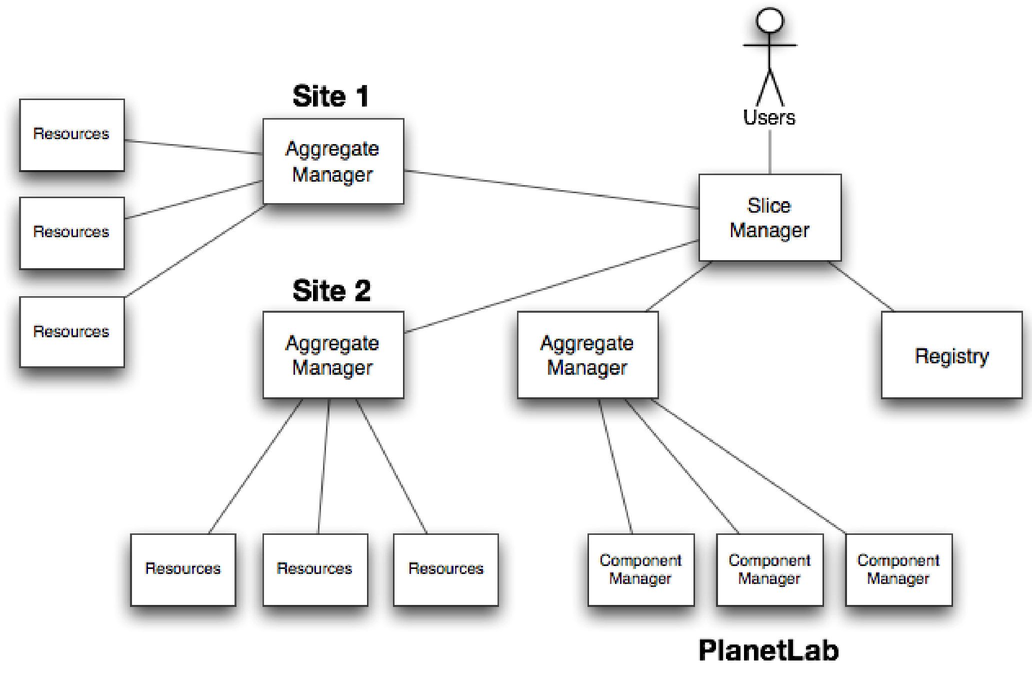

Figure 3.3: Multiple Aggregates Federation

ibilities in data will require some way to translate to a common format such that both networks can understand. For example, if the way credentials are represented in the federating network is different from how PlanetLab credentials are represented, then federating network must either change its credentials representation or implement a layer to translate the two credential representations. In the document that describes the PlanetLab's reference implementation of SFA[77], it mentions the following sites will be using SFA to federate with PlanetLab Central: PlanetLab Europe, PlanetLab Japan, PlanetLab Korea, and PlanetLab Brazil.

A site can, instead, opt for just implementing an aggregate manager, so PlanetLab can discover and utilize the federating site's resources. In this case, the slice manager and registry are not running on the federating site, but rather only on PlanetLab. The architecture of using this method is shown in Figure 3.3. Any slice related operations involve the slice manager running on PlanetLab to contact both the PlanetLab's aggregate manager and the aggregate manager running at the federating site. The federating site's aggregate manager handles the underlying protocol details on managing the site's resources. The slice manager running on PlanetLab hides the fact that the resources are located in different sites, and presents to users the those resources as if they are located at the same site. For the GENICloud project, we have employed this method and implemented an aggregate manager for Eucalyptus.

3.4 User-oriented Federation

Various federation models discussed in Section 3.3.1 all share a common property—the federation always happens on the network level. While federation on the network level is transparent to users, it also increases the overhead for the different networks in the federation. Each network will have to have the knowledge of each other within the federation; that implies various agreements, say, the usage policy, bandwidth policy and etc., need to be first agreed upon and then signed; security policies need to be updated to take into accounts of cross-domains access and possible intrusions from within the federation or even network topology changes to accommodate the other networks in the federation. Hence, adding new networks to the federation can become a really long process, and, in the end, users are the ones who suffer. Instead of having different networks agreeing on some policies, or changing their security and network configurations, GENICloud enables the federation to be deferred down to the users level, but, at the same, without having users managing all their credentials to different networks, otherwise it is not much of an improvement of the status quo. GENICloud takes the middle approach, where there is a service which acts as a proxy for the users (Figure 3.4). This user proxy manages the different credentials for the different networks that a user has access to. For example, if a user has access (accounts) to PlanetLab, Emulab, and Eucalyptus, the user proxy will manage the user's credentials for all those networks; when the user wants to create a slice that spans across those networks, the user proxy can use the credentials to access those networks, create the slice and provision the slice with resources from the different networks. The user-oriented federation reduces the overhead for network administrators; they no longer need to know about all other federating networks, and for the users they do not need to manually manage their credentials to different networks. New networks can be added to the federation with ease, because all of the modifications required are moved down from the network level to the user proxy level where it resides in between the networks and users. The user proxy can be implemented as a web service and it can be either running on a dedicated server, or it can be running within a cloud so long as users it is accessable via the Internet.

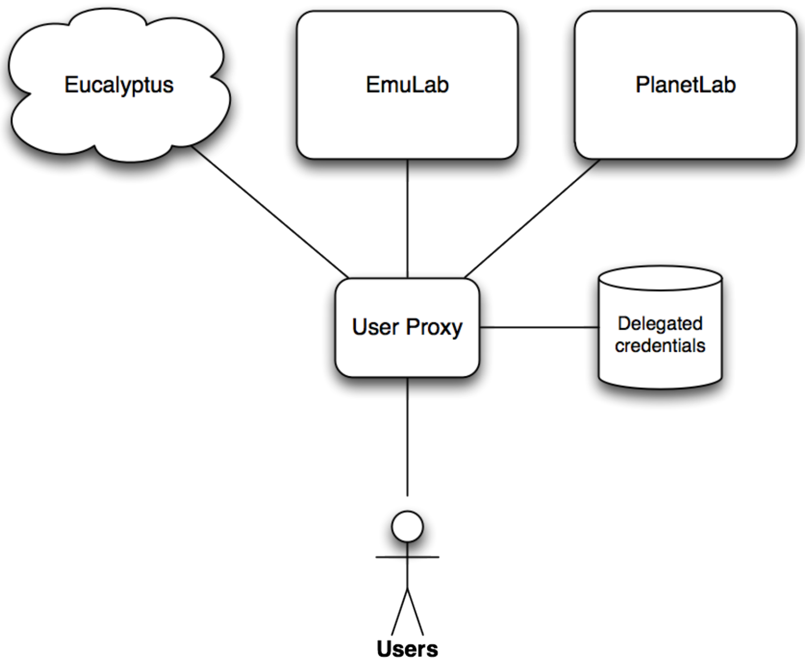

Figure 3.4: User Proxy Architecture

3.4.1 Requirements for User-oriented Federation

Most existing networks(e.g., PlanetLab, Emulab) have their own application programming interface (API), and possibily different authentication and authorization mechanisms. Such heterogeneous mechanisms and API puts a heavy burden on the user proxy, but Slice-based Facility Architecture (SFA) can mitigate the problems. SFA defines a set of common interfaces which the federating networks can implement and SFA defines the `credential` data type which can be used by the networks for authentication and authorization purposes. SFA provides the user proxy a more unified view of the federating networks. Therefore, GENICloud only supports networks that implement the SFA interface.

Networks like PlanetLab, Emulab, and Eucalyptus all use `SSH` public and private keys as a way to authenticate and authorize users, as public/private keys is one of most robust method of establishing and maintaining trust between the service providers, network testbeds, and the users[67]. The users upload their public keys to the networks through various means, typically through the network's own web interface and keep the private keys away from anyone except for themselves. If private keys are compromised, then the users' accounts on the networks will become vulnerable to unauthorized access. Since private key should **only** be in the users

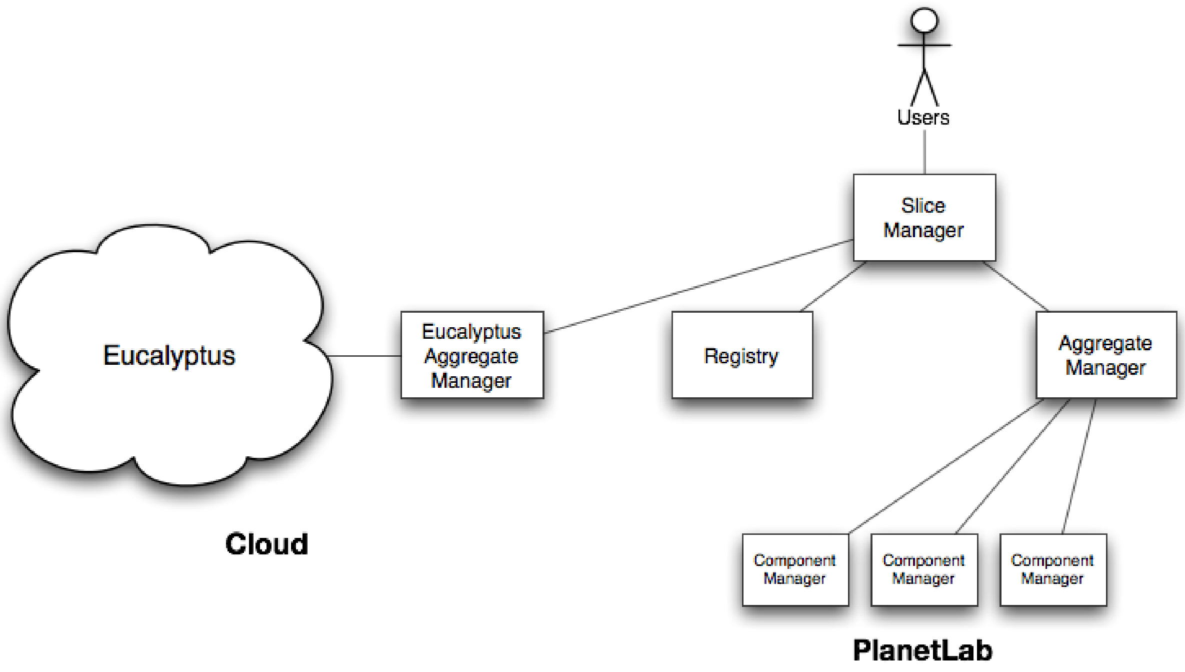

Figure 3.5: GENICloud Architecture

possession, the user proxy must avoid managing users' private keys. Instead of having the users handing out their private keys to the user proxy, it requires the users to provide the user proxy the credentials for the networks. The credential, in this case, is a X.509 certificate which contains only the user's public key not the private key and it is digitally signed by the private keys belonging to the networks[60]. To acquire the credential, the users will need to use their private keys to log into one of the networks, after which, the credentials will be created and can be used to access the network resources without using the private keys. Moreover, the users can create delegated credentials where the users just delegate certain rights to the user proxy. Delegation of rights can prevent privilege escalation and adhere to Principle of Least Privilege[81]. With the delegated credentials, the user proxy only has enough rights to perform what it has to do, and it cannot perform anything other than what the users intended. In the case where the user proxy or the private key is compromised, the users can revoke their delegated credentials any time.

3.5 Design of Eucalyptus Aggregate Manager

This section describes the overall architectural design of the Eucalyptus aggregate manager. In Section 3.3.1, I briefly mentioned that the GENICloud uses the multiple aggregates method to federate with PlanetLab. The architecture can be seen in Figure 3.5. In our design, the Eucalyptus aggregate manager is responsible for controlling and managing an Eucalyptus cloud. For example, whenever the users want to provision their slices with Eucalyptus VM instances, they will submit a RSpec specifying the type of instances they want using the `SFI` tools to the slice manager running on PlanetLab. With the right configuration, the slice manager will contact all of the aggregate managers that are known to it, and that including the Eucalyptus aggregate manager. The Eucalyptus aggregate manager will receive the users submitted RSpec from the slice manager. After parsing the RSpec, the aggregate manager will allocation the instances according to the RSpec and map those instances to the slices. The aggregate manager can be run as a standalone server. It will use the Eucalyptus API to manage Eucalyptus instances. That being said, in the GEC 7 demo, we have setup the Eucalyptus aggregate manager to run within an Eucalyptus cloud as an Eucalyptus instance and control the cloud from within. Before diving into the implementation of Eucalyptus aggregate manager, the credentials used by both systems need to be explained as dealing with credentials is the first thing the users have to face. In the next section, I will explain the interaction between the users and both systems during the authentication process. Since both systems have different ways of handling authentication as well as authorization, we have to devise a credential management model for GENICloud in order to federate Eucalyptus with PlanetLab.

3.5.1 Credentials

This section explains the authentication and authorization mechanisms used by PlanetLab and Eucalyptus. For PlanetLab, in addition to the user name and password, acquired by registering at the website, one uses to log into the web interface, the users have to upload their public keys. Their public keys are used to inject into the allocated nodes, so the users who possess the matching private keys can log into the slivers of the slices that belong to them and start running their experiments. For Eucalyptus, the credentials involved are a little more complicated than PlanetLab. Mainly because there are two sets of credentials involved that the users will use during their interaction with Eucalyptus. Similar to PlanetLab, users of Eucalyptus will have to register before they get the user name and password to access the web interface. Their registration will, then, have to be approved by the administrator of the Eucalyptus cloud. Once the users are logged into the web interface, the users will have to download their credentials which are generated automatically by Eucalyptus. The credentials are bundled as zip archives. The zip archive contains a private key, a X.509 certificate[60], an access key and a secret key. Both the access key and secret key are just string values. However, unlike PlanetLab where users upload their own public keys in order to access their slivers, Eucalyptus users have to create at least one public/private key pair using the Eucalyptus Web Services API or the Eucalyptus command line tool (e.g., `euca2ools`). However, using the Eucalyptus' API and command line tool required proper credential. Hence, the credential generated by Eucalyptus automatically, either the X.509 certificate or both access and secret keys in the zip archive, have to be used to make API calls or the command line tool. The key pair created using the API or command line tool are used to gain access (`ssh`) to the Eucalyptus instances.

3.5.2 Credentials Management

As mentioned in the previous section, PlanetLab and Eucalyptus have different ways to deal with authentication and authorization. Problems relating to credentials arise when federating Eucalyptus with PlanetLab. First, the Eucalyptus aggregate manager uses the Eucalyptus API to control an Eucalyptus cloud, but in order to use the Eucalyptus API, the aggregate manager is required to have a valid credential. Therefore, the aggregate manager will need to have some way to acquire a credential to do its jobs. Second, Eucalyptus' users cannot upload their public keys for logging into the instances. Such model does not compatible with the model which PlanetLab uses. When an Eucalyptus cloud is federated with PlanetLab, an ideal situation is that PlanetLab users can create instances on the Eucalyptus cloud, and the public keys that the users uploaded to PlanetLab should propagate to the Eucalyptus instances. Please note, although such problems can be resolved using the user proxy; it is still in the beginning of the prototyping phase, and we did not conceive the idea of user proxy until after the GEC 7 demo. So for the demo in GEC 7, we had to come up with ways to circumvent the credential problem. The next section will describe the models we devised.

Eucalyptus Aggregate Manager Credentials

To address the first problem where the Eucalyptus aggregate manager needs a credential, we have devised two models for the aggregate manager. Either the aggregate manager works as a delegate for the PlanetLab users, or the users submit their Eucalyptus credentials to the aggregate manager. When the Eucalyptus aggregate manager acting as a delegate, it will have its own Eucalyptus credential. So whenever the aggregate manager needs to use the Eucalyptus API, it can just use its own credential; all the requests from the PlanetLab's users, in this case, will be acted on behalf by the aggregate manager. From Eucalyptus point of view, it will only see the requests from the aggregate manager, but not from the PlanetLab users. The delegation method simplifies the process for PlanetLab users so they do not have to have an Eucalyptus account in order to use its resources. The aggregate manager will rely on a configuration file, when it is acting as a delegate. The configuration will contain the access and secret keys. As mentioned before, in order to use Eucalyptus' API, one needs either a X.509 certificate or the access and secret keys. The configuration file will be stored in `/etc/sfa` as a plain text file. Hence, proper file system permission should be set on the file to avoid unauthorized users from accessing its content. Another way for the aggregate manager to get a credential is for the users to submit their own credentials along with the RSpec. When the users need to create Eucalyptus instance, they will submit their access and secret keys along with the request, so the aggregate manager will use the submitted keys to call the Eucalyptus API. However, this method requires the PlanetLab users to have accounts on the federating Eucalyptus cloud, and possibly some changes to the way `SFI` works.

Credentials to access instances

The public keys users upload to PlanetLab will not automatically transfer to Eucalyptus, because Eucalyptus does not have the support for users to upload their own public keys. In this case, if the users already have an account on the Eucalyptus cloud, they can just create a public/private key pair use the Eucalyptus API, and use the key pair to log into the instances. However, if the users do not have an user account on Eucalyptus, they will have to rely on the aggregate manager to create a new key pair. If the aggregate manager is running as a delegate, as described in previous section, it will have a credential for itself, so it can use the Eucalyptus API to create a new key pair. The public key of the key pair will be embedded into the instances, and the private key can be embedded into a RSpec. Either way, the users will have to manager different sets of key pairs—the key pair they use to log into PlanetLab node, and the key pair they use to log into Eucalyptus instances. At this point, there is not a more elegant solution due to the limitations at both systems.

3.6 Fault Tolerance

As mentioned before network environments are very dynamic, hosts can come and go without any warnings. A node from PlanetLab maybe available at this instance but become unreachable in the next; on our end, GENICloud simply cannot do anything to prevent against such problems; it is simply out of GENICloud's control and even the service providers' control (e.g., underseas fiber optics links are damaged). Researchers and users should understand that in a dynamic environment anything can and will go wrong. The only way to deal with that is to anticipate the problems and handle the problems as gracefully as possible, instead of trying every which way to prevent those problems.

3.7 Summary

In Section 3.1, a comparison between Eucalyptus and PlanetLab is discussed, and one could say that there are quite a few similarities architecturally between the two. The various design decisions for different components in GENICloud hinge on those similarities. A missing architectural abstraction, a slice, in Eucalyptus is implemented by GENICloud. The users of GENICloud should also understand problems in a network environment are inevitable and the users should guard against them instead of preventing them from happening.

Chapter 4

Implementation of GENICloud

As discussed in previous chapter, I have outlined the conceptual designs and their benefits of GENICloud. Now, it is time to dive into the implementation details of GENICloud. Various components of GENICloud are implemented and running on various clusters (e.g., OpenCirrus, EmuLab), and some are still being implemented. The implementation details of the components that are implemented will be discussed in this chapter, and the ones that are being implemented will be discussed in Chapter 6. Along with implementation details, there are code snippets to provide contexts and examples.

The implementation of GENICloud uses various scripting languages including Python, JavaScript, and Bash. A few of scripting languages have grown mature enough to see their way into mainstream applications and services. Moreover, their syntax is easy to learn and they provide a lot of functionality in their standard library by default, as such scripting languages are great tools for rapid-prototyping.

This chapter will start off with explaining the implementation of Eucalyptus Aggregate Manager, one of the major components in GENICloud. Then a normal work flow on how to use the Eucalyptus aggregate manager is presented. The work flow shows what a normal user would do when using the aggregate manager, and it includes detailed examples of the commands and the explanations for the commands.

4.1 Implementation of Eucalyptus Aggregate Manager

Most of the implementation effort of GENICloud concentrated on implementing the aggregate manager on top of Eucalyptus. In addition, a resource specification format is formulated for Eucalyptus. The aggregate manager acts an mediator between PlanetLab and an Eucalyptus cloud. It manages the creation of Eucalyptus instances for slice, also it maintains a mapping of slices and instances so when the users query the sets of resources allocated for their slices, the information is readily available. This section begins with explaining what a resource specification is, and the format of the resource specification we have formulated for Eucalyptus. Then, a normal work flow of users for using Eucalyptus aggregate manager is explained. For the rest of this section, other aspects of the implementation are discussed.

4.1.1 Resource Specification (RSpec)

The resource specification plays an important role in the interaction between the Eucalyptus aggregate manager, explained in Section 3.2, and the users. The resource specification is a XML document that can be used by the aggregate manager to return information to the users and the users can then use it to send information to the aggregate manager. Since the resource specification is in XML format, the format of the RSpec for a specific network is completely open for the network to define. Having such openness nature, the RSpec can encompass many different types of resources and different network topologies. As a result, many networks (e.g., PlanetLab[76], VINI[38], ProtoGENI) have different RSpec formats. For the GENICloud project, we have defined a RSpec for Eucalyptus, so that its resources and requests from users can be expressed in XML format. During the work flow, described in Section 4.1.2, users interact with the slice manager using RSpec devised for Eucalyptus. In general, a RSpec, ignoring the different networks' specific information, should convey three types of information, depending on what operation the user is invoking. First, the RSpec should be expressive enough to inform the users of the capacity and capability of the network. In other words, the RSpec should inform the users what resources are available. Second, the RSpec should allow users to express their requirements for the resources. So, they can specify the resources they want in the RSpec, and the slice manager and aggregate manager will try to satisfy those requests. Lastly, the RSpec

```
<vm_types>
  <vm_type name="m1.small">
    <free_slots>0</free_slots>
    <max_instances>2</max_instances>
    <cores>1</cores>
    <memory unit="MB">128</memory>
    <disk_space unit="GB">2</disk_space>
  </vm_type>
  <vm_type name="c1.medium">
    <free_slots>0</free_slots>
    <max_instances>2</max_instances>
    <cores>1</cores>
    <memory unit="MB">256</memory>
    <disk_space unit="GB">5</disk_space>
  </vm_type>
  ...
</vm_types>
```

Listing 4.1: An excerpt from the RSpec showing the different types of instances in the cloud

should contain information about resources that are already provisioned to the users.

The RSpec we devised for Eucalyptus can inform the users about the resources available in an Eucalyptus cloud. Users can use the RSpec to submit their requests for resources in an Eucalyptus cloud, and inform users the instance provisioned to a slice. When the RSpec is used to inform the resources in a cloud, the contents of the RSpec contain the types of instances the users can instantiate (Listing 4.1) , the different images available for the instances (Listing 4.2) , the public keys available to be embedded in the instances (Listing 4.3) , as well as other information about the cloud and the clusters.

The RSpec for PlanetLab as shown in Listing 4.4 follows a different format because of its resources and network topology. The RSpec of PlanetLab, when listing resources, shows the sites in PlanetLab Central (PLC) and PlanetLab Europe (PLE). Within each site, the RSpec lists the nodes that belong to that site. Similar to the Eucalyptus, the users can create and assigns slivers to slices, and customize different aspects (e.g., bandwidth limit) of the slivers.

```
<images>
  <image id="emi-88760F45">
    <type>machine</type>
    <arch>x86_64</arch>
    <state>available</state>
    <location>images/ttylinux.img.manifest.xml</location>
  </image>
  <image id="eki-F26610C6">
    <type>kernel</type>
    <arch>x86_64</arch>
    <state>available</state>
    <location>images/vmlinuz-2.6.16.33-xen.manifest.xml</location>
  </image>
</images>
```

Listing 4.2: The different images (e.g., disk images, kernel images) for instances

```
<keypairs>
  <keypair>cortex</keypair>
  <keypair>mykey</keypair>
</keypairs>
```

Listing 4.3: The keypairs in the cloud

RSpec Validation

As mentioned at the beginning of this section, RSpec is used as the input parameter and output from the slice manager. As a result it is a good programming practise to validate the input parameter to protect against particular forms of attacks or prevent leaving the aggregate manager in an undetermined state when there are errors in the submitted RSpec. Whenever a RSpec is passed into the slice manager and aggregate manager, the RSpec will be validated against a schema. The schema is written in

```
<site id="s4">
  <name>Kentucky</name>
  <node id="n73">
    <hostname>planetlab1.netlab.uky.edu</hostname>
    <bw_limit units="kbps">100000</bw_limit>
  </node>
  <node id="n74">
    <hostname>planetlab2.netlab.uky.edu</hostname>
    <bw_limit units="kbps">100000</bw_limit>
  </node>
</site>
```

Listing 4.4: A snippet of PlanetLab RSpec

RELAX NG[20]. If the RSpec does not validate against the schema, the managers will not continue, and will notify the users of the error in the RSpec. A copy of the schema is bundled with the source code of SFI toolkit.

4.1.2 Users Work Flow

This section describes the work flow of a typical user. Under most circumstances, the users will follow the same work flow described in this section. At this point of the GENICloud implementation, the users are required to have accounts at PlanetLab and at an Eucalyptus cloud. Once the users acquired the credentials, they will have to download and setup the SFI command line tools, since the users will be primarily using it to interact with PlanetLab and Eucalyptus. The SFI tools support different commands. In a normal work flow, the first step involves discovering what sort of resources are available to at a network; in GENICloud, the users can find out the resources available at PlanetLab or at an Eucalyptus cloud. As such, SFI tools provide a command called `resources` for resource discovery. The command returns a RSpec (Section 4.1.1). After the resource discovery process, the users, with sufficient privileges, are free to choose resources they want to assign to the slices they own. Recall that only the Principle Investigator of a PlanetLab site can create slices; a normal PlanetLab user can become a member of slices but not create them. As mentioned before, adding resources to slices are done using the RSpec. The users will edit the RSpec with the resources they want to allocate and submit the edited RSpec using the SFI tools. What to edit in the RSpec depends on the sites; both PlanetLab and Eucalyptus expect different request formats in the RSpec. The users should be aware of the respective format when submitting the resource allocation requests. After the resources are successfully allocated to the slice, the users can query, using the SFI command line tools, about the resources that are allocated to the slice. Querying about the provisioned resources allow the users to learn more information about the those resources. For example, by querying what Eucalyptus instances are allocated to a slice, the users can learn the IP addresses of those instances and thereby allowing them to remotely log in to those instances. The query result returned by the SFI tools is a RSpec; a snippet of the RSpec containing the query result can be seen at Listing 4.5. Resources allocated to a slice can be changed. For example, the users can choose to allocate more instances to a slice or remove existing instances from the slice. In order to change the resources allocation to a slice, the users just have to edit

```
<euca_instances>
  <euca_instance id="i-3C1107C5">
    <state>running</state>
    <public_dns>155.98.36.233</public_dns>
    <keypair>cortex</keypair>
  </euca_instance>
</euca_instances>
```

Listing 4.5: Allocated instance

a RSpec with the new resources that they want, and resubmit the RSpec using the `SFI` tools.

4.1.3 Slice and Instance Mapping

One of the main abstractions for PlanetLab is the idea of slices. However, as mentioned in Section 3.1 Eucalyptus does not have any concepts of slices. In other words, Eucalyptus just treats all the instances as individual virtual machines; they are not associated with each other even though they may be related. The lack of slices in Eucalyptus does not work with PlanetLab, slice-based network, nor does it satisfy the architectural requirement of SFA. In order to provide the slice abstraction to Eucalyptus, the Eucalyptus aggregate manager has the responsibility of keeping track of the slice and instance mapping. The slice and instance mapping responsibility falls into the Eucalyptus aggregate manager because it has knowledge about both systems and it can communicate with both of them using their remote interfaces. Another way to implement the concept of slices is to modify Eucalyptus' source code, and it would mean forking the project and rendering existing Eucalyptus clouds incompatible with GENICloud. One side effect of the Eucalyptus aggregate manager is that the users of Eucalyptus now have the slice abstraction for their virtual machines; they can group related Eucalyptus instances into an single slice and manage them that way.

Internally, the aggregate manager maintains the mapping between instances and slices using a `SQLite3` database. Whenever the users add or remove instances to and from a slice, the aggregate manager keeps track of the allocation changes. The mapping is essential, especially it is used to reveal to the users information about the instances that are allocated for their slice including the instances' IP addresses, so the users can log into those instances.

Inside the `SQLite3` database, two tables are used to maintain the mapping. Listing 4.6 show the schemas for both tables. For the time being, the schemas are relatively

```
CREATE TABLE slice (
    id INTEGER PRIMARY KEY,
    slice_hrn TEXT
);
CREATE TABLE euca_instance (
    id INTEGER PRIMARY KEY,
    instance_id TEXT UNIQUE,
    kernel_id TEXT,
    image_id TEXT,
    ramdisk_id TEXT,
    inst_type TEXT,
    key_pair TEXT,
    slice_id INT CONSTRAINT slice_id_exists REFERENCES slice(id)
);
```

Listing 4.6: Schemas for the tables

simple. The `slice` table only keeps track of the human readable name of the slice (`slice_hrn`). The instance table (`euca_instance`) retains a few properties of an Eucalyptus instance; for example, the kernel image id, ramdisk id, and etc. But the most important field on the `euca_instance` table is the foreign constraint named `slice_id`; the column establishes a many-to-one relationship between slices and instances whereby a slice can be associated to many instances but not the other way around.

4.1.4 Resource Discovery and Slice Allocation

This section explains the inner workings of the aggregate manager when the users want to find out the resources available at an Eucalyptus cloud, as well as, change allocation of instances to a slice. In addition, the `SFI` command used to perform the said operations will be shown.

Resource Discovery

The users will use the `SFI` tools to discover resources available at an Eucalyptus cloud. The result will be returned to the users as a RSpec. The command `sfi.py resources` is used to query Eucalyptus for its resources. During the execution of the resource discovery, the function `get_rspec` in the aggregate manager will be invoked. Inside the function `get_rspec`, the aggregate manager will attempt to create a connection to the Eucalyptus interface. If a connection cannot be established, due to whatever reasons, the aggregate manager will stop and log the error in a log file. If the connection is established successfully, an API call is made to Eucalyptus, and the call returns the information about the clusters registered to the cloud. The clusters' information include the types of instances, number of instances can be created, and etc. The information is parsed and transformed into a RSpec. When the `sfi.py resources` is followed by a slice name (e.g., `sfi.py resources <slice hrn>`), the instances that are associated with the given slice will be returned in the RSpec. For this case, the aggregate manager has to do some extra work to find out information about the instances. Similar to the general resource discovery procedure, the `get_rspec` function is called with the given slice name passed as a parameter, and it will attempt to create a connection to the Eucalyptus API. After a connection is made, the aggregate manager will query its `SQLite3` database to look for any instances that are mapped to the slice. If mapped instances are found, the aggregate manager will collect the Eucalyptus instance ID's of those instances, then use the Eucalyptus API call to find out the instances' states, IP addresses, and etc. All the information will be embedded into a RSpec and returned to the users.

Slice Allocation

The users will have to edit the RSpec returned by `sfi.py resources`. The edited RSpec should be submitted using the command `sfi.py create <slice_name> <edited_rspec>`. In the aggregate manger, the `create_slice` function is called. Just like all other operations, a connection to the Eucalyptus API service needs to be established before proceeding forward. The slice name and the content of the RSpec is passed to the aggregate manager; the `create_slice` function in the aggregate manager will be called with the slice name and RSpec. After a connection is established, the submitted RSpec is validated against a schema. If the RSpec is deemed invalid, the aggregate manager will log the reason why the RSpec failed the validation to a log and return immediately. Depending how the users edit the RSpec, new instances could be added to a slice or instances could be removed from the slice. After validating the RSpec and the validation succeeds, the aggregate manager will parse the RSpec in order to determine what the users want in their slice. The slice's human readable name (HRN) is used an identifier for the database to find any mapped instances. If the slice is not in the database, a new record is created and all the instances that associated with that slice is also recorded. The `sfi.py create` command does not return anything to the users, but the users will be notified if the allocation was unsuccessful.

4.1.5 Python Libraries

Just like other software, GENICloud's implementation of the aggregate manager utilizes other software packages in order to avoid reinventing the wheel; also, to simplify the code base, and ease the maintenance overhead. This section outlines those packages and explain their use in the aggregate manager.

`boto` This is python interface to the Amazon Web Services. It supports many different web services that Amazon offers including Simple Storage Service (S3), Simple Queue Service (SQS) and others. But the most important interface in `boto` for the GENICloud project is the interface for Amazon Elastic Compute Cloud (EC2). Since Eucalyptus' interfaces are compatible with interfaces in EC2, so `boto` can be used with Eucalyptus. The aggregate manager uses `boto` to interface with the federating Eucalyptus cloud.

`xmlbuilder` This simple module provides a "pythonic" way of creating XML. The aggregate manager uses the `xmlbuilder` to generate the RSpec which is an XML document. The aggregate manager first uses `boto` to gather information about the Eucalyptus cloud, then it programmatically generates the RSpec using the data it gathered using `xmlbuilder`.

`sqlobject` To maintain the mapping between slices and instances, a `SQLite3` database is used, and the aggregate manager uses `sqlobject` to provides a object relation mapping between python objects and `SQLite3` tables. This module helps to eliminate the need for hand-written `SQL` statements in the source code, and keeps the implementation of aggregate manager object oriented.

`lxml` When the users submit an RSpec to the aggregate manager, the submitted RSpec needs to be validated and parsed in order to satisfy the users' requests. The `lxml` modules provides functionality to validate the RSpec against a schema, and parse RSpec so that the aggregate manager can add or remove instances from a slice. Any RSpec that fails validation will be rejected, in order to keep the aggregate manager from encountering unexpected errors.

```
—> { "method": "echo", "params": ["Hello JSON—RPC"], "id": 1}
<— { "result": "Hello JSON—RPC", "error": null, "id": 1}
```

Listing 4.7: JSON-RPC

4.2 Implementation of User Proxy

For the user proxy, the implementation involved a backend component and a frontend component. The backend component will be implemented either as a RESTful[55] web service or a RPC service using JSON or XML as the encoding format. The frontend is what the users will interact with. It will give the users a rich and dynamic experience in their web browsers. Through the web browsers, the users can interact with GENICloud in a more intuitive and convenient way compare to the command-line interface that I have demonstrated in previous sections.

For the backend, we will be using one of the most popular Python web frameworks, Django[3]. The backend will make calls to the PlanetLab implementation of SFA. Mainly, the backend will direct the requests to the slice manager which implements the slice interface (Section 3.2) defined by SFA. Django as a model-view-controller web application framework can expose the backend through various means. One way to make the backend available to the frontend is to expose a RESTful interface of the backend. The frontend can use simple HTTP[54] request methods to communication with the backend. For example, if the frontend needs to display to the users all of the resources available, the HTTP request method, `GET`, can be used. Also, if parameters need to be passed to the backend from the frontend, a HTTP `POST` can be used to pass those parameters to the backend. For example, if the users want to find out all the resources of a particular slice, one can `POST` the slice HRN to the backend. Another way for the frontend to communicate with the backend is through Remote Procedure Call[72] (RPC). RPC can marshal input data as well as return data from an application into a format that can be transmitted over the network. On the receiving end, the marshalled data will be converted back into native types within the application. There are two popular formats to marshal the data—XML and JSON[49]. Examples of the marshalling methods can be seen in Listings 4.7, and 4.8.

In the era of Web 2.0, be it a fad or it is here to stay, users are experiencing a whole new way of browsing websites or using web applications with the web browsers. One

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

Listing 4.8: XML-RPC

key technology for delivering such experience to the users is through the use of Ajax or Asynchronous JavaScript + XML[58]. As a result, the frontend will be a mix of JavaScript, and HTML[40]. Data will be asynchronously retrieved from the frontend without users intervention, and the page can be updated by manipulating the Document Object Model (DOM) using JavaScript, again without having the users manually refreshing the page. However, writing web application frontends that support all major browsers is always the bane for most web applications developers, because not all browsers support the same standards, some have their own interpretation of the standards, and some just implement their own extensions. Although JavaScript is a very nice scripting language, throughout the GENICloud project, we have been using Python. In order to avoid having to context switch between languages, we have decided to use Pyjamas[17]—a framework for developing Ajax applications in Python. Pyjamas compiles Python code into JavaScript and it has a quite extensive widgets set the we can use to build the frontend. Using Pyjamas can alleviate the overhead of dealing with cross-browsers issues, and speed up development because it provides a more programmatic way to layout widgets on the web page and it comes with a Ajax framework.

There is another feature of Pyjamas that we can take advantage of. Pyjamas has a desktop widget library which one can use to create desktop applications. The library is called Pyjamas-Desktop. With Pyjamas-Desktop, we can create a complete desktop application that wraps around `sfi.py` without having to implement the backend. So the frontend and backend are merged together and become a cross-platform desktop application.

4.3 Summary

This chapter discussed intricate implementation details of numerous components in GENICloud. The implementation, through the use of SFA, is made easy for any eager developers to pick up and contribute. Alongside with the very detailed implementation discussions, this chapter presents a straight forward work flow for normal users. The work flow should is kept very simple, so users can adopt GENICloud easily.

Chapter 5

Evaluation

Although GENICloud is still under active development, there are still certain criteria that can be used to evaluate the project developed so far. One important criterion that is not yet available is the performance metrics. GENICloud's main contributions are not any novel algorithms, so, while, performance metrics are important, other aspects of the project are equally important if not more. In this chapter, GENICloud is evaluated based on its usability, security and community impact.

5.1 Usability

User experience is one of the cornerstones of users adoption. GENICloud needs users to act as a feedback mechanisms to continue to evolve. At the same time, users need systems like GENICloud, where they can access vast amount of computing resources easily and efficiently. A system with subpar user experiences can deter the users from adopting it. Unsatisfactory user experiences can include unintuitive user interface, difficult to install and deploy, poor performance, etc. However, designing human-centered systems[41] is out of the scope of this thesis.

When evaluating the usability of GENICloud, there are three points of views to take into account. First, there is the perspective from users, which include their experiences with using GENICloud, then there is the perspective from the maintainers and administrators of the testbeds or service providers; they are ultimately what make GENICloud usable, without them there would not be any resources for provisioning. Lastly, the perspective from GENICloud's developers.

5.1.1 Users' Perspectives

Windows Millennium Edition(ME) made quite a few improvements over Windows 98, one of Microsoft most successful Windows releases. Windows ME has a better implementation of TCP/IP stack, has support for Universal Plug and Play, has a better Graphical User Interface, has a new automatic updates feature, and etc. These few improvements mentioned are only a subset of the actual new and updated features in Windows ME. As a user, I certainly find those features really attractive and useful. However, Windows ME was and is still being touted as one of the "The 25 Worst Products of All Time"[88]. The users of windows ME had been plagued by the problems such as installation problems, problems with running Windows ME, compatibility with other software and hardware, and getting ME to stop running[88].

While GENICloud may not be as big of a project as Windows ME, but it can potentially suffer the same usability problems. In Section 4.1.2, I outlined the work flow of a normal user of GENICloud. The work flow is designed to be as streamlined and straight forward as possible. Aside from the one time setup steps (e.g., acquiring credentials, create a delegated credential), the normal work flow can be represented in three simple steps—users first discover the resources available, request the resources based on their requirements, and discover the resources allocated. Under normal circumstances, the users can get their resources they need for their experiments in three steps. In this linear flow, there are no extra steps in between that will not lead them one step closer to what the users want—getting resources from GENICloud.

The work flow of GENICloud is simple and straight forward. Now, let's look at the command line interface, `sfi.py`, mentioned in Section 4.1.2. Recall that `sfi.py` or `SFI` command-line tool is the only way to interact with PlanetLab's reference implementation of SFA (Section 3.2). The way `sfi.py` works is admittedly relatively clunky, because it involves quite a lot of XML editing. Since the input parameters and output of `sfi.py` commands are all in RSpec, which essentially is a XML document. During resource discovery, the users have to first store the output, RSpec, of `sfi.py` using the `-o` option, then users edit the RSpec using an text editor. The RSpec will now contain a list of resources. For testbeds like PlanetLab where it has more than a thousand of nodes, editing such big XML document is very error prone and daunting; users cannot quite get a big picture and what type of resources are available because of the overwhelming output; even worse they may even have to manually filter through resources that are not suitable for their experiments. Ultimately, requiring the users to edit XML file makes using the `sfi.py` command line tool very error prone and hard to manage when dealing with large set of resources. Realizing the shortcomings of `sfi.py`, we are in the process of conceiving a web graphical user interface for GENICloud. The backend of the web GUI will interact with PlanetLab's implementation of SFA. Our approach to the GENICloud web GUI is either build one from scratch with various libraries and tools (Section 4.2) or modify one of the existing web GUI since other research groups have built web GUI's for similar purpose. The strong contenders are the PlanetLab existing web GUI, Emulab's web GUI or the GUI from the Seattle project[24].

5.1.2 Administrators' Perspectives

GENICloud is like a matchmaking services. It brings resources to users and vice versa. If GENICloud has plenty of users, but not enough resources, such case would be detrimental GENICloud's growth and adoptions. As a result, the easier it is for the resources' administrators and service provider to become a part of GENICloud, the more resources we can provision to users. Users always demand for more resources, and as the user base grows the demand grows as well. Being an administrator is already a demanding job. Between keeping to update with the latest security patches, and managing user accounts, the administrators stretch really thin in terms of time to deal with other issues or requests. GENICloud strives to reduce such extra burden as much as possible for the administrators when they decide to federate their resources with GENICloud.

GENICloud is always looking for computing resources. Our main target of resources are existing Eucalyptus clouds or any generic clusters where we can install Eucalyptus and turn those clusters into clouds. Once the administrators agree to federate their resources with GENICloud, they will need to install and setup the Eucalyptus aggregate manager (Section 4.1). The Eucalyptus aggregate manager allows other networks that implement the SFA interfaces to talk the Eucalyptus cloud, thereby allowing the federation to start using the resources located at the newly join Eucalyptus cloud. The Eucalyptus aggregate manager is extremely easy to setup. All of the executables are packaged as RPMs, the administrators just need to have a Fedora Core 8 image available, and install those RPMs. Better yet, the aggregate manager can even run inside the cloud on one of Eucalyptus instances. In other words, the administrators do not need to set aside a dedicated machine for the Eu-

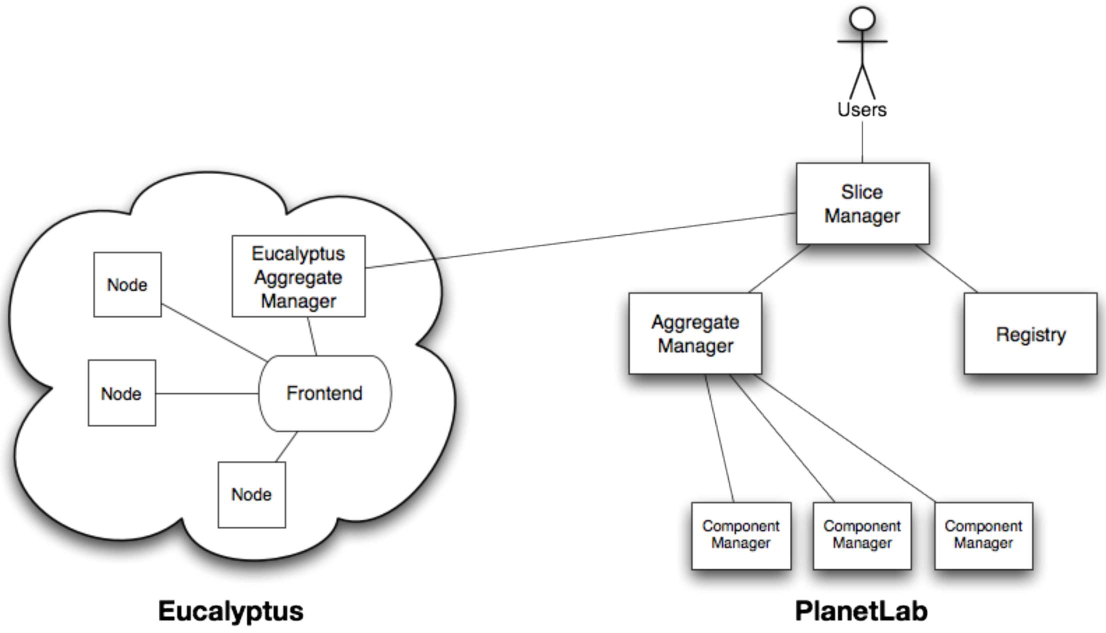

Figure 5.1: GEC 7 Demonstration

calyptus aggregate manager. If running the aggregate manager within the cloud, the administrators only need to download the disk image, boot up an instance using the downloaded image. Hence, the administrators do not even need to bother with install anything, they only need to create an instance, log in and make a few changes to the configuration file. In fact, for our demonstration at GENI Engineering Conference 7 (GEC 7), we used the aforementioned setup for our demonstrate the ability to create slices on Eucalyptus (Figure 5.1). Last piece of the component in GENICloud, the user proxy (Section 3.4), should not even need any attention from the administrators. Since the user proxy is designed to be run as a third party service. It is something we, the GENICloud developers, need to deploy and maintain.

The involvement of the administrators on the technical side has been kept to minimal. However, on the administrative side will most likely to take longer depending on how many documents the administrators need to deal with. Other overhead can include change in network topology or firewall rules to allow public access, but these only happen to generic clusters. Existing Eucalyptus clouds mostly do not have to worry about anything else but setting the Eucalyptus aggregate manager which is rather simple.

5.1.3 Developers' Perspectives

GENICloud is an open source project. Just like any other open source projects, GENICloud needs developers' contributions so it can keep growing as the project matures. New features and bug fixes are really important for community adoption, because they give life to the project. Potential adopters can see that there are developers within the project who are active and looking for feedback from the user base. Users and network administrators alike, would like to see that GENICloud is being maintained and new features requests are being accepted and fulfilled. As a result, it is very important to retain and recruit existing and new developers. In order to achieve that, we have to look at GENICloud from inside out and make sure the project is easy to join, easy to contribute and easy to develop.

Attracting new developers is one way to breath new life into a project. GENICloud welcomes any new developers who believe in what the GENICloud can bring to the research community. Joining the GENICloud project should be a simple, quick and painless process because the existing members should not spend to much time on accepting new developers and the new developers should not be deterred by lengthy joining process. Currently, the development team of GENICloud is small. If any developers who are interested are welcome to contact any one of the existing developers, including myself, or they can contact the manager. And for those who are interested and had not made a decision can join our bi-monthly conference calls. Twice a month, all the people who are involved in the GENICloud project will have a conference call and talk about the status of the project as well as the future of the project.

Another great advantage of open source projects is that anyone can contribute to the projects. In an open source project environment, contributors submit patches that fix existing or unknown bugs, or they submit patches for new features. Usually those patches are submitted via Email. Such model has been employed the Linux Kernel development for a long time, where contributors submit patches to the Linux Kernel Mailing List[13]. A quick browse to the mail list, one can spot a lot of subject lines with the `[PATCH]` prefix. Those are the patches submitted by the contributors. The patches are then picked up by Linus Torvalds at his discretion to incorporate them into the mainline kernel. GENICloud works in similar fashion, but in a miniature scale. Contributors can send in patches through email to anyone of us. In fact, the patch submission work flow is how I have been developing GENICloud. Since I do not have write access to the PlanetLab repository, I submit code of GENICloud as patches via email to another developer and he can review the code and commit the code into the PlanetLab repository. The PlanetLab code repository is publicly open for read access to anyone, so any potential contributors can get a hold of the GENICloud source code and create patches.

Minimizing the learning curve for developing GENICloud is another way to attract more developers and thereby keep the GENICloud growing. GENICloud is mostly written in Python[18]. Over the years Python has grown into one of the most used mainstream general purpose scripting languages; it has matured to the point that a lot of companies start deploying their enterprise-grade products in Python. For example, Python is one of the languages supported by Google App Engine[7], where a lot of web applications running on Google App Engine are written in Python. Python's popularity comes from its simple, easy to learn syntax and its extensive standard library. Moreover, some of the GENICloud code base is based on PlanetLab, if any new developers who are interested in GENICloud have prior experience with PlanetLab's code base, their learning curve is reduced even more. For those who have no prior experience, there are plenty of documentation one can catch up on. Since the GENICloud's developers are all geographically separated, we collaborate on the GENICloud design using Google Wave[8]. All of the design documents for GENICloud are available on Google Wave. Also, there are a lot of useful documents available regarding SFA on PlanetLab's website. With all the resources readily available on the Internet, any new developers should find the learning curve relatively conquerable, and they can start developing patches sooner.

5.2 Security

One of the main GENICloud's component, user proxy, has the important tasks of managing users credentials. Managing credentials is a very touchy subject. When most users learn that some third party component will manage their credentials for them, they will start to cringe, because they are afraid of the party component will misuse their credentials—the lack of trust between users and third party components. However, the users cannot be blamed for the lack of faith in third party components. Most modern systems still use login name and password as authentication and Access Control List (ACL) as authorization. In other words, an identity needs to be established first, usually through login name and password, and the service provider will query some internal database (ACL) to determine whether the user can access the resource. But there are plenty of drawbacks from using ACL in those systems and in distributed systems[61, 28]. In order to instill trust between users and third party components, GENICloud, followed suit with other network testbeds, uses a different model to approach the authentication and authorization problem. Instead of utilizing ACLs, GENICloud employs a trust model between users and its components.

5.2.1 Trust Management

In trust management systems[44, 43], they shun away from relying on resolving identities to establish an authorization decision[42]. Trust management systems utilize credentials (e.g., X.509), security policy, and trust relationships to determine whether a principal (user) can access the resources she requested. The security policy and trust relationships can be expressed in a programming language. The trust model is, compare to traditional ACL, is a more scalable. In the distributed systems environment users can come and go as they please; hence, managing their identities can be a real problem. Another property of trust management is being able to delegate certain rights to other principals which can decentralized the administrative structure. Moreover, individual domains can still enforce its own security policy. For example, PlanetLab can *trust* all users from an Eucalyptus cloud to use its resources, but not all users from Emulab. As such most network testbeds, PlanetLab, Emulab, and etc. all employ a trust model to make authorization or authentication decision.

5.2.2 GENICloud Trust Management

The user proxy (Section 3.4) in GENICloud is responsible for accessing different network testbeds and acquire resources for users. However, it does so without requiring the users to relinquish their login names and passwords or private keys. Instead the user proxy needs the users' credentials, but the users have the option to create a delegated credential just for the user proxy. In fact, it is recommended for the users to create delegated credentials for the user proxy. Users are establishing a trust relationship between the themselves and the user proxy, by creating a new delegated credential with only a subset of the rights. The `sfi.py` command-line tools have a `delegate` command which allows a user to delegate all or a subset of her privileges to another user (or objects). From the users point of view, they know the user proxy cannot perform any tasks that they do not want the user proxy to do because it just does not have enough privilege to do so. And, if the users do not, for whatever reasons, trust the user proxy, they have the ability to revoke the delegated credentials anytime they want.

From the networks' point of view, they do not know the user proxy exists; in fact, they should not have to know. Because of the fact that federation happens between the users and networks but not on the network level. Those networks do not need to know or manage other networks' credentials. One nice property about the user proxy is that the federated networks are only managing their own users but not users from other domains. Also, The networks can still enforce their own membership approval process. Since the user proxy only manages the credentials for the users, it is still the users' responsibility to acquire the credentials from different networks that they want to use. For example, if a user wants to use resources from PlanetLab, and Emulab, she still has to apply for accounts from both of the networks. Each network can have different approval process and usage policy, and the user will have to agree and comply to all of them. Once she gets the credentials from the networks, she can delegate those credentials to the user proxy, and the user proxy will acquire the resources she wants on her behalf. To the networks, they can still keep an audit trail, in case of usage policy violations, because the user proxy is using a delegated credential. As a result, the user can still be held accountable for their actions. The user proxy provides as much security as the networks would provide since it is using the networks' credentials, authentication and authorization procedures, and in turn those networks can to enforce their own authentication and authorization policies.

5.3 Community Impact

GENICloud is not only a tool for researchers and users to acquire new resources for their experiments and services, I believe it also enables researchers to conduct new experiments and users to test and develop their new services to further pushing the envelop of cutting edge network research. Researchers and users alike always hope for more resources especially during conference deadline or demonstration time in front of the valuable investors. Moreover, for long running services, they have to cope with unpredictable traffic spike. GENICloud provides such platform for experimenters to conveniently acquire resources from PlanetLab and any slice-enabled Eucalyptus clouds. The users and experimenters have the power to deploy services or experiments all around the world and, if needed, dynamically provision their services with more computing resources and storage capability.

5.3.1 Applicability

GENICloud's overall usability is quite users, administrators, and developers friendly (Section 5.1). But the thing that is important is what they use GENICloud for, and the context at which they use GENICloud. One of the recent trends in computing is the move from desktop applications to the Internet applications. Especially, moving exiting software and services onto a cloud. Such trend renders the users' personal computers to become just a terminal and all of the data and applications are hosted in a cloud where they can access pretty much anywhere.

In recent years, there is an emerging trend in database technology—NoSQL. Developers start to find relational database management systems (e.g., MySQL) do not scale and perform very well in an environment where the trends are moving towards web applications which can potentially have thousands or millions of users at any given time and scaling horizontally instead of vertically. New structured data storages[46, 51] start to become popular, because one of their intrinsic properties is that they can scale horizontally. By scaling horizontally, new computing resources can be added to the systems thereby increasing their throughput. Such scalability fits really with the cloud computing paradigm where it excels in on-demand dynamically provision of a shared pool of resources. Other services are staring to spring up that are based upon the cloud computing platform. For example, Heroku[11], one of the leading deployment and development platforms for Ruby web applications, is based on the cloud computing model. Users of Heroku can develop their web applications using any of the Ruby web applications frameworks (e.g., Rails) and deploy them on Heroku in a simple and coherent work flow. In other words, Heroku provides what is known as platform as a service (PaaS). Another example of a service that leverages the cloud computing paradigm is Salesforce[23]. In a nutshell, Salesforce is a company that hosts and distributes business software on a cloud and clients or businesses can subscribe those business software. Salesforce provides business software as a service (SaaS) to their clients. With all these exciting technologies and real-world applications, GENICloud provides networking researchers the benefits of cloud computing for their cutting edge research in network technology.

5.3.2 Impacts on Existing Networks

Not only just users and researchers can use GENICloud to build great services or conduct research, existing networks can also reap the benefits of cloud computing using GENICloud. Existing networks can federate with Eucalyptus clouds rather easily. As long as those networks implement the SFA interfaces, they will be able to federate their networks with other existing Eucalyptus clouds using GENICloud. The SFA interfaces are designed to be as simple and generic as possible. GENICloud makes federation relatively easy for existing networks; such benefits can be viewed both ways. It can encourage those existing networks to open up their resources and join a federation, or GENICloud can help existing networks to get more resources from Eucalyptus clouds. Since adding support to a new networks only happens in the user proxy level, the administrators only have to worry about implementing the SFA interfaces, but not all the paper works and other overhead if the federation happens on the network level. As a result, the administrators or service providers would be more willing to sharing their resources and as a side effect, they get access to more resources. For the networks which always struggling with the users' demand for more resources, they can participate in a federation thereby gaining access to more resources. As a last note, any networks that already implement the SFA interfaces, can easily use GENICloud to federate with other networks and clouds. Emulab may eventually implement those SFA interfaces since they already have very similar interfaces implemented. PlanetLab and Emulab are working towards to reconcile their interfaces.

5.4 Summary

While performance metrics are not yet available, they may not be the most import criterion for projects like GENICloud. For GENICloud, adoption is, perhaps, the most important criterion. This chapters evaluates GENICloud usability from three different perspectives. Also, the security aspect of GENICloud is very important and by employing trust management, the system is more secure. Existing testbeds, public or private, will benefit a lot from GENICloud because of how easy it is to federate with other testbeds.

Chapter 6

Future Work

GENICloud is only at its inception. In a short few months of time, we have made decent progress towards our goal. However, there is still a lot of room to grow. Completing and expanding the feature sets of GENICloud is the top priority. In the subsequent sections, I will describe a few features that are planned. Some of them are ongoing and some of them are something that we planned for the future.

6.1 Walrus Storage Service

Referring back to Figure 1.2 in Chapter 1, the Walrus storage service is part of Eucalyptus. However, at the current implementation GENICloud, the Walrus storage service is not exposed to users. The Walrus storage service provides storage to Eucalyptus users and one can think of it as the open source counterpart of Amazon Simple Storage Service (S3). In order to add support for Walrus, the first important task is to add new stanzas to current Eucalyptus' RSpec (Section 4.1.1). One of those additional stanzas will describe the capabilities of the Walrus service; for example, the RSpec should inform users the storage usage statistics (e.g., free storage left in `GB`), and the RSpec display the storage buckets that the user has access to. Walrus uses the buckets abstraction for its storage model. Users can create or remove buckets and put or delete objects (files) inside those buckets. Other additional stanzas are for users to edit. They can request buckets to be created and removed in the RSpec. Uploading and Deleting files can be done using existing command-line tools provided by Eucalyptus or using the Amazon's S3 command-line tool since Walrus is API-compatible with S3.

6.2 Web GUI/User Proxy

The `sfi.py` command-line tools, as discussed in Section 5.1.1, always require the users to edit the RSpec, which makes using the `sfi.py` command-line tools very error prone when dealing with large pool of shared resources. Having the users to manually edit XML can be error-prone, and frustrating. Hoping to improve users experience with GENICloud, there is an ongoing effort to create a web graphical user interface (GUI). Currently, we are looking into the web GUI from the Seattle project[24], hoping that we can adopt it and use it to work with PlanetLab and Eucalyptus using `SFI`. At the same time, we are also exploring the idea of creating a web GUI from scratch as it will give us more control and we have found great tools to build such web GUI[3, 17]. The users will have an aggregated view of the all resources available in the federated network—PlanetLab nodes and Eucalyptus resources. Creating and removing slices can also be done within the web GUI, as well as, provisioning those slices. So users can hand pick the PlanetLab nodes they want, and customer their Eucalyptus virtual machines using the Web GUI. If the web GUI is successful, it will potentially replace the existing ageing PlanetLab's web GUI.

6.3 Multiple Eucalyptus clusters

In the current implementation of GENICloud, the users can create slices on an Eucalyptus cloud. In the GEC 8 demonstration in July, we will demonstrate the ability to create slices across multiple Eucalyptus clouds. For such setup, multiple Eucalyptus aggregate managers will need to deploy for each of the Eucalyptus clouds participating in the federation. The architecture will be similar to what is described in Figure 3.3. There will be one slice manager, and it will have knowledge about other Eucalyptus aggregate managers. As mentioned before those Eucalyptus aggregate managers can be run within the clouds. Either each Eucalyptus cloud runs its own aggregate manager within itself, or all of aggregate managers be run within a single cloud. When the users make requests to the slice manager, it will subsequently contact all the aggregate managers. Those aggregate managers will process the request and return the results, if any, to the slice manager.

6.4 Public PlanetLab Integration

For testing and development purposes, we are using `MyPLC`, a portable version of PlanetLab Central, in place of the public PlanetLab. Since there are some features have yet implemented, and the RSpec for Eucalyptus is still being finalized, enabling GENICloud with public PlanetLab is a disaster waiting to happen. However, `MyPLC` gives us a perfect development and testing package. With `MyPLC`, we can deploy our own private PlanetLab easily and have the settings we want. But, ultimately, we would like to integration public PlanetLab into GENICloud, so that GENICloud users can use public PlanetLab's nodes and PlanetLab's users can take advantage of the dynamically provisioning capability and storage facility of Eucalyptus clouds.

Chapter 7

Conclusions

GENI is an ambitious project with a goal to promote computer network innovations and push the envelope of networking research. As part of the GENI project, there are a few existing network testbeds (e.g., PlanetLab, Emulab) providing researchers with the resources they need to conduct experiements and deploy services. Those testbeds all have a very impressive number of nodes or computing resources, but, eventually, as the GENI project grows more resources are going to be needed. Cloud computing is a perfect solution for such dynamically changing environment where demand from users can fluctuate based on various factors. There are a lot benefits from incorporating the cloud computing paradigm in GENI. One of the most popular usages of cloud computing is scaling. GENI researchers can dynamically scale up or down their experiments. Another area where GENI researchers can benefit from cloud computing is its computing and storage capability. For computing capability, the famed programming model MapReduce[50] can be deployed very easily on the cloud. MapReduce implementation like Hadoop[10] can be deployed and used by researchers to add computational power to their services or experiements[63]. For storage capability, researchers can store their data, either the result data or the experimental data, using one of the storage services provider by the cloud. In Eucalyptus, the Walrus storage service is available. Using the storage services, the researchers do not have to worry about maintaining their own disks array, backing up their data, and having to worry about the storage limit. Although it may be a lesser known usage of cloud computing, but cloud computing can facilitate communications between domains or experiements by using simple queues or enterprise service buses[82]. Cloud computing can provide very expansive and valuable services to existing users of testbeds. The GENICloud project provides a convenient way to federate Eucalyptus clouds with PlanetLab and potential other testbeds. So researchers and users can enjoy the benefits of cloud computing. Aside from federating PlanetLab with Eucalyptus clouds, GENICloud also provides an easy way to federate other SFA-based networks all through the use of user proxy. Using GENICloud, the users can acquire and manage computing resources from different networks and clouds through a single interface and a straight forward work flow. Through federating Eucalyptus and PlanetLab, GENICloud identifies a few architectural design decisions that are shared between the two systems and other testbeds. Of course, there are some differences as well; for example, the lack of slice abstraction in Eucalyptus, but the Eucalyptus aggregate manager in GENICloud fills in that gap. By identifying those architectural similarities, GENICloud provides a blueprint for next generation testbeds.

Bibliography

[1] Amazon elastic compute cloud. Website, 2010. http://aws.amazon.com/ec2.

[2] CERN. Website, July 2010. http://www.cern.ch.

[3] Django. Website, 2010. http://www.djangoproject.com/.

[4] Emanicslab. Website, 2010. http://www.emanicslab.org/.

[5] FlexiScale. Website, 2010. http://www.flexiant.com/products/flexiscale/.

[6] GENI. Website, 2010. http://www.geni.net/.

[7] Google app engine. Website, 2010. http://code.google.com/appengine/.

[8] Google Wave. Website, 2010. http://wave.google.com/.

[9] Gush: GENI user shell. Website, 2010. http://gush.cs.williams.edu/trac/gush.

[10] Hadoop. Website, 2010. http://hadoop.apache.org/.

[11] Heroku. Website, 2010. http://heroku.com/.

[12] Kernel-based Virtual Machine. Website, 2010. http://www.linux-kvm.org.

[13] Linux kernel mailing list. Website, 2010. http://lkml.org/.

[14] Linux-VServer. Website, 2010. http://linux-vserver.org/.

[15] OneLab. Website, 2010. http://www.onelab.eu.

[16] Planetlab Japan. Website, 2010. http://www.planet-lab.jp/.

[17] Pyjamas. Website, 2010. `http://code.google.com/p/pyjamas/`.

[18] Python. Website, 2010. `http://www.python.org/`.

[19] Rackspace. Website, 2010. `http://www.rackspace.com`.

[20] Relax NG. Website, 2010. `http://relaxng.org/spec-20011203.html`.

[21] RightScale. Website, 2010. `http://www.rightscale.com/`.

[22] Rocks distribution. Website, 2010. `http://www.rocksclusters.org/wordpress/`.

[23] Salesforce. Website, 2010. `http://www.salesforce.com/`.

[24] Seattle. Website, 2010. `https://seattle.cs.washington.edu/html/`.

[25] Shibboleth. Website, 2010. `http://shibboleth.internet2.edu/`.

[26] Ubuntu enterprise cloud. Website, 2010. `http://www.ubuntu.com/cloud/private`.

[27] K. Aamodt et al. The ALICE experiment at the CERN LHC. *JINST*, 3:S08002, 2008.

[28] Martín Abadi, Andrew Birrell, and Ted Wobber. Access control in a world of software diversity. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems*, pages 127–132, 2005.

[29] R. Adolphi et al. The CMS experiment at the CERN LHC. *JINST*, 3:S08004, 2008.

[30] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, Berlin, 2004.

[31] David G. Andersen, Hari Balakrishnan, Frans M. Kaashoek, and Robert Morris. Resilient overlay networks. In *Symposium on Operating Systems Principles*, pages 131–145, 2001.

[32] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.

[33] JF Arguin. Atlas experiment status and first results. Technical Report ATL-GEN-PROC-2010-004. ATL-COM-GEN-2010-018, CERN, Geneva, May 2010.

[34] Ilia Baldine, Jeff Chase, George Rouskas, and Rudra Dutta. At-scale experimentation with resource virtualization in a metro optical testbed. In *International Conference on the Virtual Computing Initiative (ICVCI 2008)*, 2008.

[35] Paul Barham, Boris Dragovic, Keir Fraser, Steven H, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP*, pages 164–177, 2003.

[36] Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, Monte Goode, and Kate Keahey. Identity federation and attribute-based authorization through the globus toolkit. In *Shibboleth, GridShib, and MyProxy. In Proceedings of the 5th Annual PKI R&D Workshop*, 2005.

[37] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[38] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In vini veritas: realistic and controlled network experimentation. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–14, New York, NY, USA, 2006. ACM.

[39] Boualem Benatallah and Hamid R. Motahari Nezhad. Service oriented architecture: Overview and directions. pages 116–130, 2008.

[40] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0. RFC 1866 (Historic), November 1995. Obsoleted by RFC 2854.

[41] Hugh Beyer and Karen Holtzblatt. *Contextual design: defining customer-centered systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[42] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210. Springer-Verlag.

[43] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Infrastructures (Position Paper). Lecture Notes in Computer Science 1550*, pages 59–63, 1998.

[44] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.

[45] M Bozzo, J Bourotte, Maurice Haguenauer, G Sanguinetti, Giorgio Matthiae, and J Velasco. Total cross section, elastic scattering and diffraction dissociation at lhc: Expressions of interest not presented at the meeting. page 2 p, Mar 1992.

[46] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th conference on USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, pages 205–218, 2006.

[47] Jeff Chase, Ionut Constandache, Azbayer Demberel, Laura Grit, Varun Marupadi, Matt Sayler, and Aydan Yumerefendi. Controlling dynamic guests in a virtual computing utility. In *International Conference on the Virtual Computing Initiative (ICVCI 2008)*, May 2008.

[48] Jeff Chase, Laura Grit, David Irwin, Varun Marupadi, Piyush Shivam, and Aydan Yumerefendi. Beyond virtual data centers: Toward an open resource control architecture. In *International Conference on the Virtual Computing Initiative (ICVCI 2007)*, May 2007.

[49] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[50] Jeffrey Dean, Sanjay Ghemawat, and Google Inc. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. USENIX Association, 2004.

[51] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[52] Hans Eriksson. Mbone: the multicast backbone. *Commun. ACM*, 37(8):54–60, August 1994.

[53] Lyndon Evans. The large hadron collider. *New Journal of Physics*, 9(9):335, 2007.

[54] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785.

[55] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[56] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pages 2–13, 2005.

[57] Ian Foster and Carl Kesselman. Computational grids. pages 15–51, 1999.

[58] Jesse James Garrett. Ajax: A new approach to web applications. Website, 2010. `http://www.adaptivepath.com/ideas/essays/archives/000385.php`.

[59] Marc Goodner, Maryann Hondo, Anthony Nadalin, Michael McIntosh, and Don Schmidt. Understanding ws-federation. Technical report, IBM Corp. and Microsoft Corp., 2007.

[60] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459: Internet X.509 public key infrastructure certificate and CRL profile, January 1999. Status: PROPOSED STANDARD.

[61] Vincent C. Hu, David F. Ferraiolo, and D. Richard Kuhn. Assessment of access control systems. Technical report, National Institute of Standards and Technology, 2006.

[62] Mark Huang and Thierry Parmentelat. MyPLC user guide. Website, 2010. `https://svn.planet-lab.org/wiki/MyPLCUserGuide`.

[63] Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards optimizing hadoop provisioning in the cloud. HotCloud Workshop, 2009.

[64] Alan H. Karp. Authorization-based access control for the services oriented architecture. In *C5 '06: Proceedings of the Fourth International Conference on Creating, Connecting and Collaborating through Computing*, pages 160–167, Washington, DC, USA, 2006. IEEE Computer Society.

[65] Alan H. Karp, Harry Haury, and Michael H. Davis. From abac to zbac: The evolution of access control models. Technical report, HP Laboratories Palo Alto, 2009.

[66] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@home-massively distributed computing for seti. *Computing in Science & Engineering*, 3(1):78–83, 2001.

[67] D. Richard Kuhn and Vincent C. Hu. Introduction to public key technology and the federal pki infrastructure. In *Special Publication 800-32, 26 February 2001. Shirey Informational [Page 361] 4949 Internet Security Glossary, Version 2*, 2001.

[68] Jun Li and Alan H. Karp. Access control for the services oriented architecture. In *SWS '07: Proceedings of the 2007 ACM workshop on Secure web services*, pages 9–17, New York, NY, USA, 2007. ACM.

[69] Jun Li and Alan H. Karp. Zebra copy: A reference implementation of federated access management. Technical report, HP Laboratories Palo Alto, 2007.

[70] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB Journal*, 12(1):59–85, 2003.

[71] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. Optimizing network virtualization in xen. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 2–2, Berkeley, CA, USA, 2006. USENIX Association.

[72] Sun Microsystems. RPC: Remote Procedure Call Protocol specification: Version 2. RFC 1057 (Informational), June 1988.

[73] Akihiro Nakao, Ryota Ozaki, and Yuji Nishida. Corelab: an emerging network testbed employing hosted virtual machine monitor. In *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–6, New York, NY, USA, 2008. ACM.

[74] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:124–131, 2009.

[75] Xinming Ou, Anna Squicciarini, Sebastien Goasguen, and Elisa Bertino. Authorization strategies for virtualized environments in grid computing systems. In *Proceedings of WSSS06, IEEE Workshop on Web Service Security*, 2006.

[76] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[77] Larry Peterson, Soner Sevinc, Scott Baker, Tony Mack, Reid Moran, and Faiyaz Ahmed. Planetlab implementation of the slice-based facility architecture. Technical report, 2009.

[78] Larry Peterson, Soner Sevinc, Jay Lepreau, Robert Ricci, John Wroclawski, Ted Faber, Stephen Schwab, and Scott Baker. Slice-based facility architecure. Technical report, Princeton University, 2008-2010.

[79] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. In *SOSP '73: Proceedings of the fourth ACM symposium on Operating system principles*, volume 7, New York, NY, USA, October 1973. ACM Press.

[80] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. Grid federation: An economy based, scalable distributed resource management system for large-scale resource coupling. Technical report, Grid Computing and Distributed Systems Laboratory, University of Melbourne, 2004.

[81] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, June 2005.

[82] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The enterprise service bus: making service-oriented architecture real. *IBM Syst. J.*, 44(4):781–797, 2005.

[83] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. *SIGOPS Oper. Syst. Rev.*, 41(3):275–287, 2007.

[84] James P.G. Sterbenz, Deep Medhi, Byrav Ramamurthy, Caterina Scoglio, David Hutchison, Bernhard Plattner, Tricha Anjali, Andrew Scott, Cort Buffington, Gregory E. Monaco, Don Gruenbacher, Rick McMullen, Justin P. Rohrer, John Sherrell, Pragatheeswaran Angu, Ramkumar Cherukuri, Haiyang Qian, and Nidhi Tare. The great plains environment for network innovation (GpENI): A programmable testbed for future internet architecture research. In *Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, Berlin, Germany, May 18–20 2010.

[85] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, Frans M. Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, February 2003.

[86] Christopher Strachey. Time sharing in large fast computers. In *International Conference on Information Processing*, pages 336–341. UNESCO, June 1959.

[87] Phuoc Tran-Gia, Anja Feldmann, Ralf Steinmetz, Jörg Eberspächer, Martina Zitterbart, Paul Müller, and Hans Schotten. G-lab white paper (in german), 1 2009.

[88] Dan Tynan. The 25 worst tech products of all time. Website, 2010. `http://tinyurl.com/worst25`.

[89] Von Welch, Tom Barton, Kate Keahey, and Frank Siebenlist. Attributes, anonymity, and access: Shibboleth and globus integration to facilitate grid collaboration. In *In 4th Annual PKI R&D Workshop*, 2005.

[90] Von Welch, Jarek Gawor, Carl Kesselman, Sam Meder, and Laura Pearlman. Security for grid services. In *Twelfth International Symposium on High Performance Distributed Computing*, pages 48–57. IEEE Press, 2003.

[91] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

[92] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.

[93] H. Zimmermann. OSI reference model–the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.