Eucalyptus : A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems

UCSB Computer Science Technical Report Number 2008-10

Daniel Nurmi, Rich Wolski, Chris Grzegorczyk
Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov

Computer Science Department
University of California, Santa Barbara
Santa Barbara, California 93106

Abstract

Utility computing, elastic computing, and cloud computing are all terms that refer to the concept of dynamically provisioning processing time and storage space from a ubiquitous "cloud" of computational resources. Such systems allow users to acquire and release the resources on demand and provide ready access to data from processing elements, while relegating the physical location and exact parameters of the resources. Over the past few years, such systems have become increasingly popular, but nearly all current cloud computing offerings are either proprietary or depend upon software infrastructure that is invisible to the research community.

In this work, we present Eucalyptus, an open-source software implementation of cloud computing that utilizes compute resources that are typically available to researchers, such as clusters and workstation farms. In order to foster community research exploration of cloud computing systems, the design of Eucalyptus emphasizes modularity, allowing researchers to experiment with their own security, scalability, scheduling, and interface implementations. In this paper, we outline the design of Eucalyptus, describe our own implementations of the modular system components, and provide results from experiments that measure performance and scalability of an Eucalyptus installation currently deployed for public use.

The main contribution of our work is the presentation of the first research-oriented open-source cloud computing system focused on enabling methodical investigations into the programming, administration, and deployment of systems exploring this novel distributed computing model.

1 Introduction

Scalable Internet services [1, 4, 24, 44] deliver massive amounts of computing power (in aggregate) on demand to large, internationally distributed user communities through well-defined software interfaces. Until recently, however, access to these services has been restricted to human-oriented and simple query-style application programming interfaces (APIs). With few exceptions, an application programmer wishing to incorporate such a service as a software component had little ability to direct and control computation *inside* the service explicitly.

Cloud computing [11, 46] has emerged as a new paradigm for providing programmatic access to scalable Internet service venues. [1] While significant debate continues with regard to the "optimal" level of abstraction that such programmatic interfaces should support (c.f., software-as-a-service versus platform-

[1] The term "cloud computing" is considered by some to be synonymous with the terms "elastic computing," "utility computing," and occasionally "grid computing." For the purposes of this paper, we will use the term "cloud computing" to refer to cloud, elastic, or utility computing but not to grid computing. The difference is explained in Section 4.

as-a-service versus infrastructure-as-a-service [13, 25, 26, 34]), the general goal is to provide users with the ability to program resources within a very-large-scale resource "cloud" so that they can take advantage of the potential performance, cost, and reliability benefits that access to scale makes possible.

In short, the model is to provide a large user base with the ability to program some specified fraction of the resources hosted by a scalable service provider (e.g., Google [24], Amazon [4], SalesForce [44], 3Tera [1], etc.) through one or more well defined service interfaces. However, while the interfaces are public, the infrastructure maintained by the various service providers is almost exclusively proprietary. Thus it is not possible (or at least not easy) for researchers to build, deploy, modify, instrument, or experiment with a cloud infrastructure under their own control.

In this paper, we describe the design and implementation of Eucalyptus – an open-source software infrastructure architected specifically to support cloud computing research and infrastructure development. The design of Eucalyptus is distinctive in that it

- must be able to deploy and execute in hardware and software environments not under the control of its designers, and
- must be modularized to allow component-wise modification or replacement,

while achieving the greatest degree of scalability possible. This work describes the system architectural trade-offs imposed upon the design by these two requirements, the way in which they have been addressed by the current version of Eucalyptus that is currently available and in use, and the degree to which these trade-offs impact the functionality and performance of the overall system.

The motivation for Eucalyptus is an exploratory one. Cloud computing as an emerging concept has great potential, but the speed of commercial engineering leaves fundamental questions either not fully defined or unanswered. Thus, while cloud systems are providing users a valuable service, the closed nature of the software has created a situation where researchers interested in cloud computing topics are finding it difficult to formulate experiments due to the lack of a common, flexible framework in which they can work.

1.1 Open-Source Infrastructure as a Service

Although most existing cloud computing implementations share the common high-level notion of flexible, scalable, and dynamic computational "provisioning," there is significant variation in exactly how that power is presented to the end user. Some systems, such as Amazon's Elastic Compute Cloud (EC2) [17] and Enomalism [18], allow users to allocate entire virtual machines (VMs) on demand, thus providing what is commonly referred to as Infrastructure as a Service (IaaS). Here, the user is responsible for providing the operating system kernel, base OS software, and any user level software and applications they wish to run and the IaaS system provisions physical resources and instantiates the user's VMs.

Eucalyptus implements IaaS, with the key differentiations being that it is specifically designed to be easy to install and maintain in a research setting, *and* that it is easy to modify, instrument, and extend. Specifically, commercial cloud infrastructures take advantage of the ability to control the local resource configuration (hardware versioning, O.S. versioning, network and storage policies, etc.) and access to large collections of potentially expensive resources (e.g., publicly visible and routable Internet addresses). In a research setting, it is unlikely that the cloud infrastructure can mandate a specific configuration for all hardware and software it manages, nor is it possible to predicate functionality on the availability of very large resource sets.

Further, because IaaS systems each typically target a specific installation, they are not engineered with extensibility or portability as a primary concern, nor is the need for ease of system administration given primacy in the design. The difficulties are compounded by the need to be able to incorporate multiple compute clusters into a single resource pool from which cloud allocations are to be drawn. Few open-source software packages of any kind are designed to install and deploy on multiple compute clusters that then operate together as an ensemble. Thus, Eucalyptus is a relatively unique example of IaaS and also a harbinger of future multi-cluster open-source design experiences. The way in which it frames and then addresses the challenges that arise as a result, forms the basis of the contribution this paper makes.

Specifically, we describe

- a simple open architecture for implementing cloud functionality at the IaaS level,
- experiences with implementing this architecture using open-source Web-service software as the intrinsic technology, and
- performance results demonstrating the viability of the resulting cloud computing system.

IaaS, however, is not the only approach to implementing cloud computing that the commercial sector is currently pursuing. Amazon and Google also both provide Data as a Service (DaaS) capabilities, through the Simple Storage Service (S3) [43] and parts of App Engine [7] respectively, where users can both store and access massive amounts of data from the provided computational resources. In addition, Google's App Engine [7], also provides a language-level abstraction, making it generically categorizable as Platform as a Service (PaaS), where access to computational power and storage is gained through language-specific APIs and libraries. Finally, companies such as Salesforce.com [44] provide a number of high-level software service packages (e.g., Web-accessible Customer Relationship Management, Enterprise Resource Planning, Inventory Control, Payroll, etc.). This higher-level approach is often described as Software as a Service (SaaS).

We have chosen to focus Eucalyptus at the IaaS level for two reasons. First, Amazon.com's EC2 is perhaps the most commercially successful cloud computing endeavor to date and it implements IaaS. Eucalyptus is interface-compatible with EC2, making it possible to test its functionality against one of the most mature commercial examples of cloud computing. This availability of a "gold standard" greatly influenced the design since it is possible to gauge immediately how closely our open-source rendition of the functionality matches its exemplar. Second, higher-level cloud computing abstractions all seem to depend on similar IaaS functionality, at least conceptually. We do not claim that all cloud computing infrastructures include an IaaS layer in their software architecture. However, for the purposes of further research and open-source development, we speculate that self-contained IaaS functionality that can be layered upon will prove both foundational and beneficial.

Further, we believe that the results garnered from our experiences with Eucalyptus may prove to be seminal. The software infrastructure in various package forms has been publicly available since approximately June 1st of 2008 and since its public release, uptake has been surprisingly rapid (so rapid, in fact, that the project has been the subject of increasingly visible discussion in the popular press, hence our decision to obfuscate its name in this paper). We believe the initial success of the project stems from our choice of EC2 as an interface to support, but also from a significant effort to make the software as easy to download and install as possible. Indeed, using the Rocks [42] cluster configuration system, installation and launch is essentially a "one-button" operation (installation from RedHat Package Management format or source is more complicated but still streamlined and documented). No other cloud system, of which we are aware, combines support for open development with ease of installation and maintenance as basic design goals while, at the same time, attempting to emulate commercially available functionality as a way of stimulating community research and development.

2 Eucalyptus Design

The Eucalyptus design is primarily motivated by two engineering goals: extensibility and non-intrusiveness. Eucalyptus is extensible as a result of its simple organization and modular design. Further, we have implemented Eucalyptus using open-source Web-service technologies, which serve to illuminate its internals. As a collection of Web services, Eucalyptus components have well defined interfaces (described by WSDL documents), support secure communication (using WS-Security policies), and rely upon industry-standard Web-services software packages (Axis2, Apache, and Rampart). This choice of implementation technology also supports the second design goal – that of non-intrusive or "overlay" deployment. We do not assume that researchers interested in Eucalyptus are necessarily willing to dedicate entire collections of machines to Eucalyptus alone (although this model of operation is also supported), nor do we assume that they are willing to allow Eucalyptus to modify the local software configuration in potentially disruptive ways. Intrusiveness is admittedly a

subjective metric. For the purposes of our work, we assume that a site wishing to use Eucalyptus is willing to support virtualized execution through Xen [8] and to host Web services. With these two requirements fulfilled, Eucalyptus can be deployed and executed without modification to the underlying infrastructure.

2.1 Architectural Overview

Academic research groups have access to a number of resources; for instance, small clusters, pools of workstations, and various server/desktop machines. Since public IP addresses are usually scarce, and the security ramifications of allowing complete access from the public Internet can be daunting, system administrators commonly deploy clusters as pools of "worker" machines on private, unroutable networks with a single "head node" responsible for routing traffic between the worker pool and a public network. Although this configuration provides security while using a minimum of publicly routable addresses, it usually means that, while most machines can initiate connections to external hosts, external hosts cannot typically connect to machines running within each cluster.

For example, an administrator might configure two small Linux clusters, a small server pool, and a collection of computer lab workstations. The clusters each have a single front-end machine with a publicly accessible IP address, while the nodes are connected via a private network such that they can only contact each other and their respective front-ends. The server and workstation machines have public IP addresses, but the workstations are behind a firewall and can not be contacted from the outside world. In this scenario, it is clear that it is not possible to install a fully connected system, since many of the machines can only initiate connections to external hosts or are entirely isolated from external networks. In addition, the two sets of cluster nodes may even have overlapping IP addresses since their networks are fully private and unroutable. In order to make all of these types of resources part of a single cloud, we reflect the hierarchical nature of this typical configuration in the architecture of Eucalyptus, as depicted in Figure 1, where the three hierarchical levels are shown. These hierarchical components are sufficiently general to accommodate installation on common network hierarchies found within many institutions, an example of which is depicted in Figure 2.

Node Controller

The Node Controller (NC) is the component that executes on the physical resources that host VM instances and is responsible for instance start up, inspection, shutdown, and cleanup. There are typically many NCs in a Eucalyptus installation, but only one NC needs to execute per physical machine, since a single NC can manage multiple virtual machine instances on a single machine. The NC interface is described via a WSDL document that defines the instance data structure and instance control operations that the NC supports (*runInstance, describeInstance, terminateInstance, describeResource* and *startNetwork*). The run, describe, and terminate operations on an instance perform minimal system setup, followed by calls to the underlying hypervisor (Xen in the current implementation) to control and inspect running instances. The *describeResource* operation reports current physical resource characteristics (compute cores, memory, and disk capacity) to the caller and the *startNetwork* operation sets up and configures the virtual Ethernet overlay described in more detail in Section 2.2.

Cluster Contoller

A collection of NCs that logically belong together report to a single Cluster Controller (CC) that typically executes on a cluster head node or server that has access to both private and public networks. The CC is responsible for gathering state information from its collection of NCs, scheduling incoming VM instance execution requests to individual NCs, and managing the configuration of public and private instance networks. The WSDL that describes the CC interface is similar to the NC interface, except that each operation is plural instead of singular (*runInstances, describeInstances, terminateInstances, describeResources*). The describe and terminate instance control operations are merely pass-thru operations to the relevant NC module. When a CC receives a *runInstances* request, it performs a simple scheduling task of determining which NCs can support the incoming instance by querying each NC through *describeResource* and choosing the

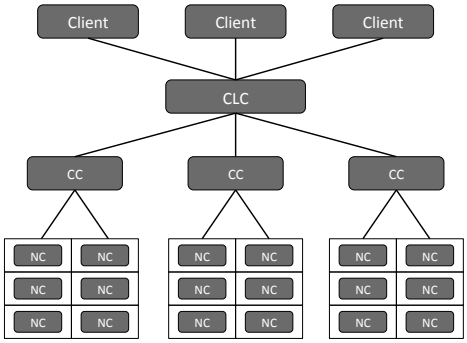

Figure 1. Eucalyptus employs a hierarchical design to reflect underlying resource topologies.

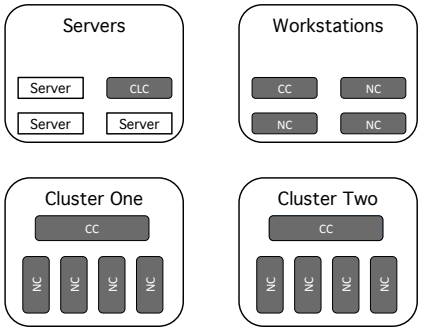

Figure 2. Example location of CLC, CC and NC components running within a typical resource environment.

first NC that has enough free resources. The CC also implements a *describeResources* operation, however, instead of reporting actual physical resources available, this operation takes as input a description of resources that a single instance could occupy, and returns the number of instances of that type can be simultaneously executed on the NCs.

Cloud Controller

Each Eucalyptus installation includes a single Cloud Controller (CLC) that is the user-visible entry point and global decision-making component of an Eucalyptus installation. The CLC is responsible for processing incoming user-initiated or administrative requests, making high-level VM instance scheduling decisions, processing service-level agreements (SLAs) and maintaining persistent system and user metadata.

The CLC itself is composed of a collection of *services* (Figure 3) that handle user requests and authentication, persistent system and user metadata (e.g., VM images and ssh key pairs), and the management and monitoring of VM instances. The services are configured and managed by an enterprise service bus (ESB) [45] that publishes services and mediates handling of user requests while decoupling the service implementation from message routing and transport details. Our design emphasizes transparency and simplicity in order to foster experimentation and extension of Eucalyptus, particularly with respect to cloud behavior. To achieve extensibility at this level of granularity, the architectural components of the CLC (including, but not limited to the VM scheduler, SLA engine, and user/administrative interfaces) are mutually isolated behind well-defined internal interfaces where ESB configuration controls their orchestration. With this as a foundation, our CLC implementation can function as an Amazon EC2 work-alike by inter-operating with the EC2 client tools using both Web-services and Query interfaces (Amazon publishes specification documents describing these interfaces). We chose EC2 because it is relatively mature, has a large existing user community, and because it implements a well-defined IaaS functionality. However, the interface parsing is modularized so that Eucalyptus can support different interfaces, either as a way of emulating other infrastructures or to allow interface customization.

Client Interface

The CLC's client interface service essentially acts as a translator between the internal Eucalyptus system interfaces (i.e., the NC and CC instance control interfaces) and some defined external client interface. For example, Amazon provides a WSDL document that describes a Web-service SOAP-based client interface to their service as well as a document describing an HTTP Query-based interface, both of which can be translated by the CLC user interface service into Eu-

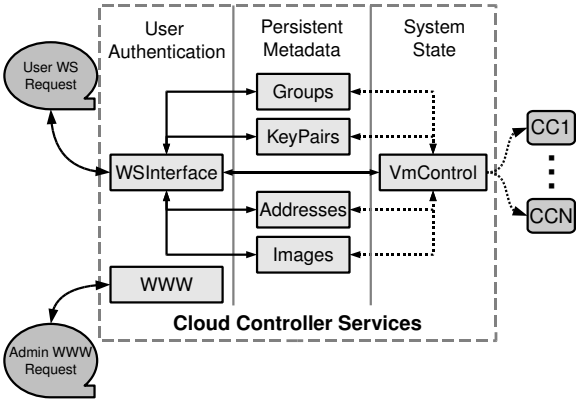

Figure 3. Overview of services that comprise the Cloud Controller. Lines indicate the flow of messages where the dashed lines correspond to internal service messages.

calyptus internal objects. We use JiBX [32] binding tool to specify a mapping of XML elements onto instances of Java objects, which we have used to create bindings that map the body of EC2 SOAP messages onto internal Eucalyptus objects.

The Query interface does not lend itself to this model however. First, there is no XML document to consume. Second, the authentication mechanism is different and in conflict with the WS-Security policy enforced. Third, conflicts exist between the structure of SOAP requests and Query requests for the same field of the same kind of request.

The solution stems from the observation that the Query interface for EC2 is a strict subset of the SOAP interface. As a result, we have developed a simple binding framework that maps HTTP Parameter names onto object fields guided by annotations. We then rely on annotations of the target object to aid in deobfuscating inconsistencies such as elided lists and unwrapped complex types (i.e., field names of a child class). Ultimately, JiBX is used to marshal the bound object using the namespace for the EC2 SOAP interface. The result is two-fold: First, JiBX will validate the object that is actually a legal SOAP interface request, hence, a legal EC2 client request. Second, the marshalled XML document can be supplied as the SOAP body to allow further processing to continue along the *exact same* path it would have taken if the message had been SOAP to begin with.

Administrative Interface

In addition to supporting primary tasks, such as starting and stopping instances, a cloud infrastructure must support administrative tasks, such as adding and removing users and disk images. Eucalyptus supports such tasks though a Web-based interface, implemented by the cloud controller, and command-line tools. Unlike the client interface, however, the administrative interface is unique to Eucalyptus. That is, while cloud purveyors do publish their client interfaces they do not generally publish administrators' interfaces. Thus, we have defined one for the system that is independent of any specific client interface or intrinsic IaaS functionality.

Users are added to a Eucalyptus installation either through the action of an administrator or by filling out an on-line form that is sent to the administrator for approval. Control over account creation thus rests in the hands of a human being, which we found necessary given the absence of automated approval methods, such as credit card verification used by Amazon. It is up to the cloud administrator to try to ensure that a new account will not be misused. By forcing new users to confirm their interest in the account by clicking on a link received in an email message, Eucalyptus maps the identity of a user to the their email address.

If that is not sufficient, the administrator may choose to verify the identity of the applicant with the help of other information on the sign-up form. Once added, a user account can be temporarily disabled or permanently removed by an administrator. At any point, the administrator can find out which instances a user is executing and terminate them.

Currently, disk images in Eucalyptus can be added to the system only by an administrator. An image consists of a Xen-compatible guest OS kernel, a root file system image, and, optionally, a RAM disk image. Adding an image constitutes uploading these three components into the system and naming the image. After a image is added, any user can run instances of that image. Administrators may temporarily disable or permanently remove the image. Finally, the administrator is in charge of adding and removing nodes from cluster controller's configuration.

Instance Control

Creation of virtual machine instance metadata in Eucalyptus is managed by a component of the CLC named the VmControl service. VmControl continuously maintains a simple local representation of the state of underlying resources (i.e., number of instances each CC could potentially create). When instance creation events are initiated, it coordinates with the other services in the CLC to resolve user request references to image, keypairs, networks, and security groups. Allocation then consists of validating references to metadata, application of an allocation strategy producing a 'pre-allocation', meaning that as far as the VmControl component is concerned, the resources have been locally reserved. Messages are then disseminated to the CCs involved in the allocation. Each such CC will schedule the instance request to its locally controlled NCs which, finally, create the virtual machine instance itself and respond accordingly.

SLA Implementation and Management

Service-level agreements (SLAs) are implemented as extensions to the message handling service which can inspect, modify, and reject the message, as well as the state stored by VmControl. Ultimately, the VmControl rationally arbitrates access to resources and enforces system-wide or user-specific service-level agreements. These decisions require data about the state of resources that is captured in a system model and the result of update events (i.e., either a change to the model or information about a failure). We have implemented an extensible SLA scheme, which couples the state model with event handling to support further work in quantitative study of service level agreements.

The VmControl relies on a local model for decision-making purposes. To keep the model up to date, each CC is passively polled to obtain the state of its instance availability, allocations, virtual network, and registered images. Information gathered via polling is treated as *ground truth* and user requests are handled in transactions that commit only when they are reflected on the resources.

Nonetheless, the model may become inconsistent, causing the system to agree to an SLA with a user that is unsatisfiable. This can happen when when messages are lost (e.g., due to network partition) and the state of resources changes (the period between polling events can be thought of as a network partition). However, loss of messages can be identified (polling is semi-synchronous) and times when the model is in an invalid state can, ultimately, also be detected (after the system recovers and ground truth can be inspected). Consequently, the likelihood that the model will be incorrect at a given moment can be computed.

We have implemented a simple yet powerful initial SLA that allows users to control the high-level network topology of their instances. While resource providers typically think of collections of machines in terms of "clusters" or "pools", we have adopted the more general concept of "zones" that is currently used by Amazon EC2. Within EC2, a "zone" is correlated to a vague geographic location such as "east coast U.S." or "west coast U.S.", while we use the term to refer to a logical collection of machines that has several NC components and a single CC component. Eucalyptus allows users to specify a zone configuration upon instance execution, which allows an instance set to reside within a single cluster or potentially across clusters. Each configuration offers different administrative and network performance characteristics, which we explore in more detail in Section 2.2. In addition, Eucalyptus further co-opts the notion of zone, extending it to support different SLAs with re-

spect to trade-offs between the number of resources acquired and their relative topology. In the current implementation, the default set of zones supplied allows users to request a specific cluster, the emptiest cluster, any single cluster unless no cluster provides the minimum requested, and multiple clusters.

2.2 Virtual Networking

Perhaps one of the most interesting challenges in the design of a cloud computing infrastructure is that of VM instance interconnectivity. One of the most attractive characteristics of cloud systems stems from the fact that although the underlying physical machines may have complex and restrictive networking topologies, a simpler, more configurable VM interconnection topology can be presented to the user through virtualization. When designing Eucalyptus, we recognized that the VM instance network solution must address *connectivity*, *isolation*, and *performance*.

First and foremost, every virtual machine that Eucalyptus controls must have network connectivity to each other, and at least partially to the public Internet (we use the word "partially" to denote that at least one VM instance in a "set" of instances must be exposed externally so that the instance set owner can log in and interact with their instances). Because users are granted super-user access to their provisioned VMs, they may have super-user access to the underlying network interfaces. This ability can cause security concerns, in that, without care, a VM instance user may have the ability to acquire system IP or MAC addresses and cause interference on the system network. In addition, if two instances are running on one physical machine, a user of one VM may have the ability to snoop and influence network packets belonging to another. Thus, in a cloud shared by different users, VMs belonging to a single cloud allocation must be able to communicate, but VMs belonging to separate allocations must be isolated. Note that current hypervisor offerings do not support this notion of grouping directly. Finally, one of the primary reasons that virtualization technologies are just now gaining such popularity is that the performance overhead of virtualization has diminished significantly over the past few years, including the cost of virtualized network interfaces. Our design attempts to maintain inter-VM network performance as close to native as possible.

Each instance controlled by Eucalyptus is given two virtual network interfaces; one is referred to as "public" while the other is termed "private". The public interface is assigned the role of handling communication outside of a given set of VM instances, or between instances within the same availability zone as defined by the SLA. For example, in an environment that has available public IP addresses, they may be assigned to VM instances at instance boot time, allowing communication both to and from the instance. In environments where instances are connected to a private network with a router that supports external communication through network address translation (NAT), the public interface may be assigned a valid private address giving it access to systems outside the local network through the NAT-enabled router. The instance's private interface, however, is used only for inter-VM communication across zones, handling the situation where two VM instances are running inside separate private networks (zones) but need to communicate with one another. The basic instance networking configuration is shown in Figure 4, which depicts the instance's public interface as connected to the public network via a bridge connected to the resource's real interface.

Within Eucalyptus, the cluster controller currently handles the set up and tear down of instance virtual network interfaces. The CC can be configured to set up the public interface network in three ways corresponding to three common environments we currently support. The first configuration instructs Eucalyptus to attach the VM's public interface directly to a software Ethernet bridge connected to the real physical machine's network, allowing the administrator to handle VM network DHCP requests the same way they handle regular DHCP requests. The second configuration allows the administrator to define a dynamic pool of IP addresses that will be assigned via a DHCP server that is executed by the CC. In this configuration, the administrator defines a network, an interface on the CC that is connected to that network, and a range of IP addresses that are dynamically assigned as instances are started. Finally, we support a configuration that allows an administrator to define static Media Access Control (MAC) and IP address tuples. In this mode, each new instance created by the system is assigned

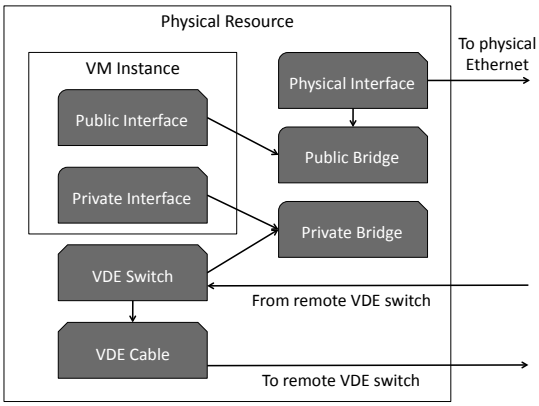

Figure 4. Each Eucalyptus VM instance is assigned a public interface for external network connections, and a private network interface connected to a fully virtual Ethernet network for inter-VM communication.

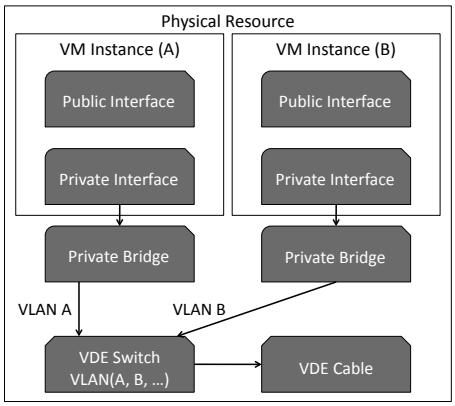

Figure 5. If two different user instances (A and B) are running on the same resource, we employ VLAN tagging as a means of isolating network traffic between VMs.

a free MAC/IP tuple, which is released when the instance is terminated.

The instance's private interface is connected via a bridge to a fully virtual software Ethernet system called Virtual Distributed Ethernet (VDE) [49]. VDE is a process-level implementation of the Ethernet protocol, where users can specify and control virtual Ethernet *switch* and *cable* abstractions that are implemented as programs running in user-space. Once a VDE network has been created, connections to real Ethernet networks can be established through the Universal TUN/TAP interface, which, in essence, provides Ethernet packet communication from the Linux kernel to user-space processes. When an Eucalyptus system is initiated, it sets up a VDE network overlay that consists of one VDE switch per CC and NC component and as many VDE wire processes as can be established between switches. If there are no firewalls existing on the physical network, the VDE network will be fully connected, where each VDE switch is connected to every other VDE switch. The VDE switches support a spanning tree protocol, which allows redundant links to exist while preventing loops in the network, thus giving the VDE network a level of redundancy when the switches are fully connected. However, since NC components may be behind a firewall, the only requirement is that each VDE switch has at least one wire to some other VDE switch in the system, which is typically satisfied by a single connection to the CC.

At instance run time, the NC responsible for controlling the VM creates a new Ethernet bridge that is connected to the local VDE switch and configures the instance to attach its private interface to the new bridge. At this point, our original requirement of instance connectivity is satisfied, since any VM started on any VDE-connected NC will be able to contact any other VM over the virtual Ethernet, regardless of the underlying physical network configuration. Currently, we allow the administrator to define a class-B IP subnet that is to be used by instances connected to the private network, and each new instance is assigned a dynamic IP address from within the specified subnet.

The second requirement of the virtual network is that it supports instance network traffic isolation. We require that if two instances, owned by separate users, are running on the same host or on different hosts connected to the same physical Ethernet, they do not have the ability to inspect or modify each other's network traffic. To meet this requirement, each set of instances owned by a particular user is assigned a tag that is then used as a virtual local area network (VLAN) identifier assigned to that user's instances. Once a VLAN iden-

tifier has been assigned, all VDE switch ports that are connected to the instance's private interfaces are configured to tag all incoming traffic with the VLAN tag and to only forward packets that have the same VLAN tag. Hence, a set of instances will only be forwarded traffic on VDE ports that other instances in the set are attached to, and all traffic they generate will be tagged with a VLAN identifier at the virtual switch level, thus isolating instance network traffic even when two instances are running on the same physical resource. Figure 5 shows how two instances owned by user A and user B running on the same physical resource are connected to the VDE network through ports configured to only forward traffic based on a particular VM's assigned VLAN.

3 Experiment

To illustrate the performance characteristics of Eucalyptus as well as to observe its functionality under user load not generated by the development team, we have installed Eucalyptus on a small research Linux cluster at our home institution and made it available for general use to the wider community as a "public cloud." The hardware configuration comprises 7 compute nodes and one head-node. The compute nodes are on an isolated network, while the front-end is publicly accessible. Each system has two Intel Xeon 3.2GHz processors, 3GB of RAM and approximately 40GB of available disk (single SCSI drive). We are running a single CLC on the front-end, a single CC on the front-end, and one NC per compute node.

Users request access to the Eucalyptus Public Cloud (OPC) by requesting credentials from the CLC through the user signup web page. Subsequent cloud allocation requests are limited to 4 instances which will be terminated automatically after 6 hours. A reverse firewall prevents EPC hosted instances from making network connections to external network addresses (public Linux distribution sites are excepted to allow instance configuration) to avoid inadvertent "spam-bot" hosting. Only a local EPC zone is available as an externally accessible SLA.

All experiments detailed in this section have been conducted using the EPC in the presence of ambient induced load. That is, unless otherwise indicated, we measured the performance of the EPC in the presence of load being generated by its users (i.e. in a non-dedicated mode).

3.1 Instance Throughput

The first experiment we perform is designed to measure the performance of VM instance control operations. Because Eucalyptus is interface compatible with Amazon's EC2, we are able to perform the same experiments on both Eucalyptus and EC2 without customization. The primary purpose in doing so is to verify that the EC2 functionality is, indeed, fully replicated by Eucalyptus. Less rigorously, the quantitative comparison serves as a high-level test for whether our implementation is pathologically inefficient. In fact, during early phases of Eucalyptus design we discovered a number of performance "bugs" through comparisons with EC2.

Because one of the primary functions of Eucalyptus is to control the execution of VM instances on a collection of resources, we perform an "instance throughput" experiment where we measure the time from when a user wishes to execute a collection of instances to the time the instances are booted and available for use on the network. For this experiment, we measure the total time between an instance execution request to the point when we can first detect that the instance is running. In order to measure the instance state, we rely on the Amazon EC2 command-line tool "ec2-describe-instances", which simply queries the cloud server for information about a user's instances and prints the information to the user's terminal. To gather a single data point, we first take a timestamp followed immediately by a launch of an instance or set of instances using the client tool "ec2-run-instances". Then, we repeatedly poll the server using "ec2-describe-instances" until our initiated instance enters a "running" state, at which point we take another timestamp. The difference between the two timestamps constitutes a single data point that represents the number of seconds between a user instance creation request and the user becoming aware that the instance(s) are available for use. Each trial is characterized by four variables and timings are reported in seconds. The first variable is the VM type requested, where the VM type is defined as the number of cores, amount of RAM, and allocated disk space. The second variable is the instance image

itself, which we control by loading identical copies inside both EC2 and the EPC. For this experiment we run trials for a "small" VM type with a corresponding image called "ttylinux" [48]; a compact Linux image that boots very quickly and offers a minimal networked Linux installation when fully booted. The third variable of interest is the number of instances simultaneously requested, which we vary from one to eight instances. The final variable is of course the system used; either Eucalyptus or Amazon EC2.

In Figure 6 we show the results of the instance throughput experiment. The figure shows two empirical cumulative distribution functions that allow us to examine both the magnitude and variance of time taken to create instances in EC2 and the EPC. Each data point represents the percentage (Y axis) of instance creation trials that took at least the number of seconds denoted at the point's corresponding position on the X axis.

Notice that for both cases (one and eight concurrent instance creation trials), though the range of creation times overlap, all empirical quantiles in the EPC case are lower than those of EC2. For example, 98 percent of the eight concurrent instance creation trials completed in less than 24 seconds within EPC, while only 75 percent completed in less than 24 seconds within EC2. For the one instance case, the difference is even more striking, with 98 percent of the trials completing in less than 17 seconds within the EPC and only 32 percent completing in less than 17 seconds within EC2.

This result, we believe, indicates that the Eucalyptus implementation is relatively efficient given its target environment. However it does not indicate that the EPC is outperforming EC2. The actual EC2 harnesses a vast resource pool and, as such, should almost certainly incur a measurable performance overhead over an Eucalyptus implementation running on a small cluster. At the same time, the implementation of the EPC does seem to compare well with that of the system it emulates indicating that its implementation is, at least, relatively high performance. This supposition is further supported by the (somewhat surprising) similarity in the shapes of two distribution plots. Both are unimodal with relatively similar tail weights. At present we are unable to go beyond this observation and to make a direct inference (say from confidence bounds on the variance) about the similarity of the two performance profiles however doing so is something we hope to achieve in the near future.

3.2 Network Performance

Our second performance experiment is designed to study the characteristics of our network solution and to compare it with EC2's networking approach. Since we do not know the network configuration and the hardware employed by EC2, it is not possible to compare the two systems in terms of functional detail. Rather large discrepancies in performance should be interpreted as indicating a significant difference in approach or a "bug" in the Eucalyptus implementation.

The network experiment was conducted between two simultaneously launched instances, one acting as a server and the other as a client. We launched the instances from a disk image of Debian's "etch" Linux distribution and installed a network performance measuring tool "iperf" into it. In addition to "iperf," which was used for TCP and UDP experiments, we used "ping" to measure the round-trip latency of an ICMP echo. Finally, the TCP buffer conditioning and experiment duration was chosen to saturate a dedicated gigabit interconnection network.

To study how physical distance between the machines hosting VMs affects network performance, we conducted this experiment both between instances within one availability zone and between instances located in two different zones. For EC2 this meant that the network traffic traveled from one Amazon site to another, albeit both located on the east coast of the US (only three availability zones are currently offered by Amazon and all three are located on the east coast). For EPC this meant that the network traffic traversed the "private" network interface implemented by VDE, as described in Section 2.2. Although instances in two different Eucalyptus zones might be able to communicate over their "public" interfaces (if all addresses are publicly routable) and thus achieve better performance, the "private" interface experiments shows how Eucalyptus performs when offering the same privacy guarantees as EC2.

The results of our network experiment are shown in Figure 7. Here, we show TCP throughput and round-trip latency between two instances, inside EC2 and

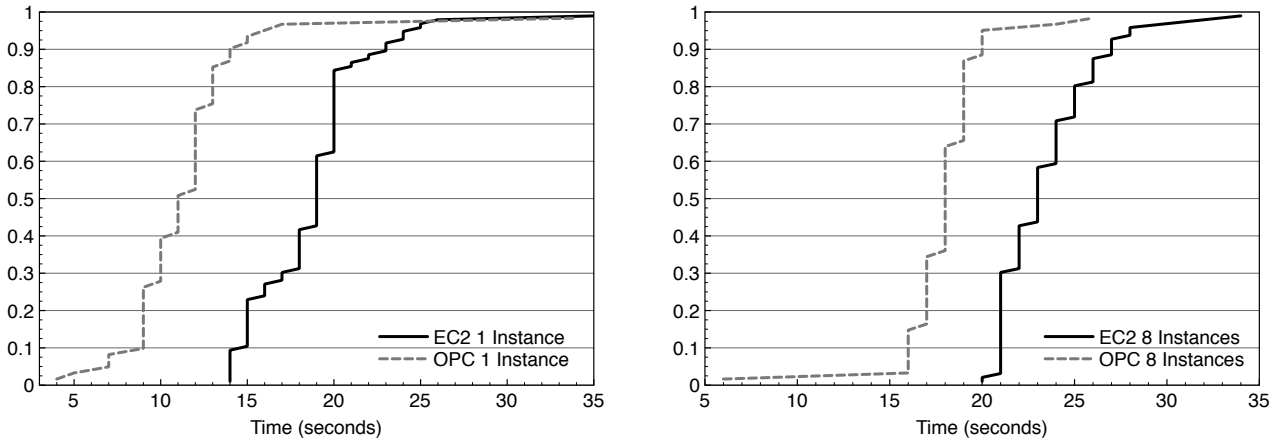

Figure 6. Empirical CDFs comparing number of seconds taken to start one and eight VM instances within EC2 and Eucalyptus.

EPC, within a single availability zone and between zones. In addition to the individual measurements from 32 independent trials, we show their arithmetic means (square features) and 95% confidence intervals, when the interval is wide enough to be of interest.

Within EC2, we observe that bandwidth within a single availability zone outperforms bandwidth between zones by approximately a factor of 2, whereas the factor is closer to 10 within the EPC. It is clear that our chosen private networking solution (VDE) imposes a significant performance penalty that is not apparent within EC2. We believe the reason for this difference lies in the fact that with VDE, the private networking overlay is running almost entirely in user space, resulting in more memory copies per packet. Another reason for the poor performance of our private networking solution can be inferred from the latency results, showing a significantly greater variance in the RTT for ICMP packets traveling over the VDE network. Similar to the bandwidth result, we found the latency in some cases was over 10 times greater between EPC zones than within a single zone, again indicating that the VDE network imposes significant network performance degradation.

In both bandwidth and latency experiments, however, Eucalyptus delivers native network performance when VDE is not selected. Thus by choosing and SLA that specifies an allocation should not span clusters, a user can ensure that her cloud allocation will realize native interconnect speed at the possible expense of scalability (since an allocation will be limited to, at most, the size of one cluster).

This experiment also demonstrates, in rather stark terms, the performance impact associated with the overlay approach implemented by Eucalyptus. Specifically, VDE is necessary to implement a secure, user-space layer-2 overlay for each cloud allocation that can span separate private cluster networks.

4 Related Work

Cloud computing stems from recent innovations in operating system virtualization and scalable Internet services. It also shares intellectual underpinning with grid computing, although the precise nature of this sharing is a matter of some debate.

Machine virtualization projects producing VM hypervisor software [8, 9, 30, 50] have enabled new mechanisms for providing resources to users. In particular, these efforts have influenced hardware design [3, 27, 31] to support transparent operating system hosting. The "right" virtualization architecture remains an open field of study [2]): analyzing, optimizing, and understanding the performance of virtualized systems [28, 29, 36, 37, 51] is an active area of research. Eucalyptus implements a cloud computing "operating system" using Xen-based virtualization as its initial target hypervisor and this work, particularly

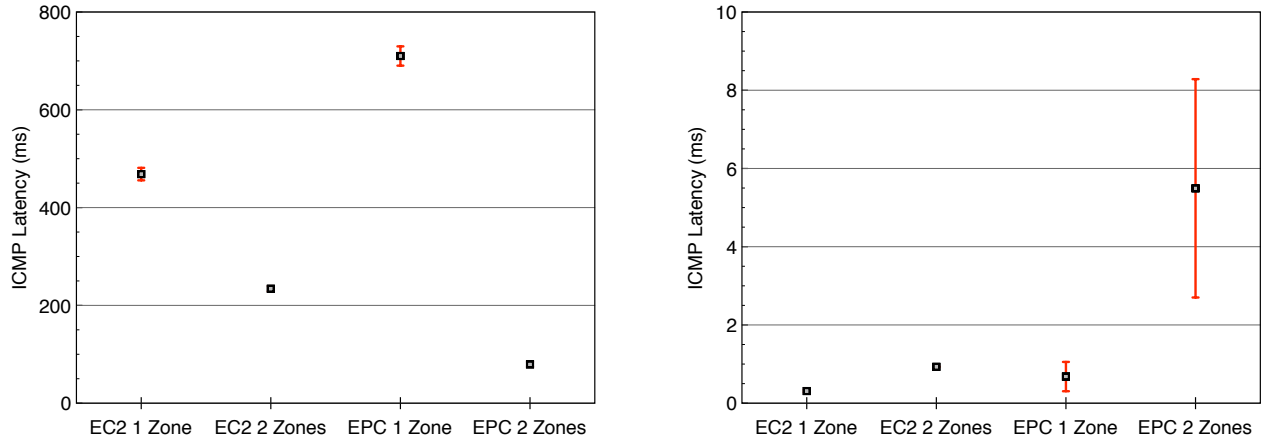

Figure 7. TCP throughput (left) and round-trip latency (right) measurements between instances started within EC2 and Eucalyptus. Individual measurements from 32 independent runs are shown together with their arithmetic means and 95% confidence intervals.

with respect to performance benchmarking, serves as a starting point for studying the overheads introduced by Eucalyptus.

Thanks in part to the new facilities provided by virtualization platforms, a large number of systems have been built using these technologies for providing scalable Internet services [1, 5, 12, 15, 16, 24, 44], that share in common many system characteristics: they must be able to rapidly scale up and down as workload fluctuates, support a large number of users requiring resources "on-demand", and provide stable access to provided resources over the public Internet. While the details of the underlying resource architectures on which these systems operate are not commonly published, Eucalyptus is almost certainly shares some architectural features with these systems due to shared objectives and design goals.

In addition to the commercial cloud computing offerings mentioned above (Amazon EC2/S3, Google AppEngine, Salesforce.com, etc.), which maintain a proprietary infrastructure with open interfaces, there are open-source projects aimed at resource provisioning with the help of virtualization. Usher [35] is a modular open-source virtual machine management framework from academia. Enomalism [18] is an open-source cloud software infrastructure from a startup company. Virtual Workspaces [33] is a Globus-based [19] system for provisioning workspaces (i.e., VMs), which leverages several pre-existing solutions developed in the grid computing arena. The Cluster-on-demand [14] project focuses on the provisioning of virtual machines for scientific computing applications. oVirt [40] is a Web-based virtual machine management console.

While these projects produced software artifacts that are similar to Eucalyptus, there are several differences. First, Eucalyptus was designed from the ground up to be as easy to install and as non-intrusive as possible, without requiring sites to dedicate resources to it exclusively (one can even install it on a laptop for experimentation.) Second, the Eucalyptus software framework is highly modular, with industry-standard, language-agnostic communication mechanisms, which we hope will encourage third-party extensions to the system and community development. Third, the external interface to Eucalyptus is based on an already popular API developed by Amazon. Finally, Eucalyptus is unique among the open-source offerings in providing a virtual network overlay that both isolates network traffic of different users and allows two or more clusters to appear to belong to the same Local Area Network (LAN).

Grid computing must also be acknowledged as an intellectual sibling of, if not ancestor to, cloud computing [10, 20, 38, 47]. The original metaphor for a computational utility, in fact, gives grid computing

its name. While grid computing and cloud computing share a services oriented approach [21, 22] and may appeal to some of the same users (e.g., researchers and analysts performing loosely-coupled parallel computations), they differ in two key ways. First, grid systems are architected so that individual user requests can (and should) consume large fractions of the total resource pool [39]. Cloud systems often limit the size of an individual request to be tiny fraction of the total available capacity [6] and, instead, focus on scaling to support large numbers of users.

A second key difference concerns federation. From its inception, grid computing took a middleware-based approach as a way of promoting resource federation among cooperating, but separate, administrative domains [19]. Cloud service venues, to date, are unfederated. That is, a cloud system is typically operated by a single (potentially large) entity with the administrative authority to mandate uniform configuration, scheduling policies, etc. Eucalyptus conforms to the design constraints governing cloud systems.

Several research projects and white papers in the last few years have studied the performance ramifications of deploying specific workloads (often scientific ones) in today's commercial clouds. For example, Palankar et al. [41] benchmarked Amazon's *S3* cloud storage solution for scientific applications, pointing out several current characteristics of the system that need to be addressed before it is appropriate for access to scientific data. Garfinkel [23] analyzed Amazon's EC2 management, performance and security facilities and reported on their experience with moving large scale research application to the cloud. This work, while valuable by itself, could be significantly augmented through experimentation with Eucalyptus, both in terms of experimental verification and by allowing the researchers of these works to more precisely understand the measured resource performance response through system instrumentation. In addition, the performance results presented in this paper are directly relevant to these other benchmarking efforts.

Overall, we find that there are a great number of cloud computing systems in design and operation today that expose interfaces to proprietary and closed software and resources, a smaller number of open-source cloud computing offerings that typically require substantial effort and/or dedication of resources in order to use, and no system antecedent to Eucalyptus that has been designed specifically with support academic exploration and community involvement as fundamental design goals.

5 Conclusion and Future Work

In this work, we have presented the Eucalyptus open-source cloud computing software framework. We have shown that Eucalyptus is distinctive among other cloud computing IaaS systems in that it supports an industry standard interface (Amazon EC2), deploys as an overlay atop existing commonly encountered resource configurations (small clusters, workstation pools, etc), and has been designed as a modular system where components may be replaced or enhanced in order to foster future cloud computing research efforts. The entire Eucalyptus system is available for download and has been successfully installed both on clusters and numerous personal computing environments.

Benchmarking Eucalyptus against EC2 reveals that it is relatively efficient. While it outperforms EC2 in absolute terms, it does so in an environment with significantly fewer resources. Only when a process-level virtual network overlay is employed is performance substantially degraded. However, by adapting the concept of availability zone from EC2, Eucalyptus allows users to trade network performance for scalability explicitly though a default set of SLAs supplied with the system. This adaptation supports the claim that Eucalyptus allows new cloud computing techniques and policies to be developed. Thus we conclude that Eucalyptus is not inherently inefficient and provides facilities for cloud computing research that are otherwise unavailable.

In addition to constantly supporting new features, we are particularly interested in using Eucalyptus as a platform for experimenting with novel cloud computing concepts such as dynamic SLA generation, new virtual networking topologies for floating static IP addresses across clouds, investigations on how to implement a truly secure cloud infrastructure, and investigating novel user and administrative cloud interfaces.

References

[1] 3Tera home page. http://www.3tera.com/.

[2] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM.

[3] Advanced Micro Devices, AMD Inc. AMD Virtualization Codenamed "Pacifica" Technology, Secure Virtual Machine Architecture Reference Manual. May 2005.

[4] Amazon.com home page. http://www.amazon.com/.

[5] Amazon Web Services home page. http://aws.amazon.com/.

[6] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[7] Google appengine – http://code.google.com/appengine/.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[9] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[10] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley and Sons, 2003.

[11] R. Buyya, C. S. Yeo, and S. Venugopa. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos, CA, USA) 2008*.

[12] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A Distributed Storage System for Structured Data. *Proceedings of 7th Symposium on Operating System Design and Implementation(OSDI)*, page 205218, 2006.

[13] M. Chang, J. He, and E. Castro-Leon. Service-orientation in the computing infrastructure. In *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering*, pages 27–33, Washington, DC, USA, 2006. IEEE Computer Society.

[14] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle. Dynamic virtual clusters in a grid site manager. *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 90–100, 2003.

[15] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Proceedings of 6th Symposium on Operating System Design and Implementation(OSDI)*, pages 137–150, 2004.

[16] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220, 2007.

[17] Amazon elastic compute cloud – http://aws.amazon.com/ec2/.

[18] Enomalism elastic computing infrastructure. http://www.enomaly.com.

[19] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.

[20] I. Foster and C. Kesselman, editors. *The Grid – Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[21] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[22] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[23] S. L. Garnkel. An evaluation of amazons grid computing services: Ec2, s3 and sqs. Technical Report TR-08-07, Center for Research on Computation and Society, School for Engineering and Applied Sciences, Harvard University, August 2007.

[24] Google – http://www.google.com/.

[25] D. Greschler and T. Mangan. Networking lessons in delivering 'software as a service': part i. *Int. J. Netw. Manag.*, 12(5):317–321, 2002.

[26] D. Greschler and T. Mangan. Networking lessons in delivering 'software as a service': part ii. *Int. J. Netw. Manag.*, 12(6):339–345, 2002.

[27] R. Hiremane. Intel Virtualization Technology for Directed I/O (Intel VT-d). *Technology@Intel Magazine*, 4(10), May 2007.

[28] W. Huang, M. Koop, Q. Gao, and D. Panda. Virtual machine aware communication libraries for high performance computing. In *Proceedings of Supercomputing 2007*.

[29] W. Huang, J. Liu, B. Abali, and D. K. Panda. A case for high performance computing with virtual machines. In *ICS '06: Proceedings of the 20th annual*

international conference on Supercomputing, pages 125–134, New York, NY, USA, 2006. ACM.

[30] Hyper-v home page – http://www.microsoft.com/hyperv.

[31] Intel. Enhanced Virtualization on Intel Architecture-based Servers. *Intel Solutions White Paper*, March 2005.

[32] JiBX home page. http://jibx.sourceforge.net/.

[33] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13(4):265–275, 2005.

[34] P. Laplante, J. Zhang, and J. Voas. What's in a name? distinguishing between saas and soa. *IT Professional*, 10(3):46–50, May-June 2008.

[35] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: An Extensible Framework for Managing Clusters of Virtual Machines. In *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, November 2007.

[36] A. Menon, A. Cox, and W. Zwaenepoel. Optimizing Network Virtualization in Xen. *Proc. USENIX Annual Technical Conference (USENIX 2006)*, pages 15–28, 2006.

[37] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis. Virtualization for high-performance computing. *SIGOPS Oper. Syst. Rev.*, 40(2):8–11, 2006.

[38] NSF TeraGrid Project. http://www.teragrid.org/.

[39] J. P. Ostriker and M. L. Norman. Cosmology of the early universe viewed through the new infrastructure. *Commun. ACM*, 40(11):84–94, 1997.

[40] oVirt home page. http://ovirt.org/.

[41] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon s3 for science grids: a viable solution? In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64, New York, NY, USA, 2008. ACM.

[42] P. Papadopoulos, M. Katz, and G. Bruno. NPACI Rocks: tools and techniques for easily deploying manageable Linux clusters. *Concurrency and Computation: Practice & Experience*, 15(7):707–725, 2003.

[43] Amazon simple storage service – http://aws.amazon.com/s3/.

[44] Salesforce Customer Relationships Management (CRM) system. http://www.salesforce.com/.

[45] M. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The Enterprise Service Bus: Making service-oriented architecture real. *IBM Systems Journal*, 44(4):781–797, 2005.

[46] D. Skillicorn. The case for datacentric grids. *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 247–251, 2002.

[47] T. Tannenbaum and M. Litzkow. The condor distributed processing system. *Dr. Dobbs Journal*, February 1995.

[48] Ttylinux home page – http://www.minimalinux.org/ttylinux/.

[49] Virtual distributed ethernet (vde) home page – http://vde.sourceforge.net/.

[50] Vmware home page – http://www.vmware.com.

[51] L. Youseff, K. Seymour, H. You, J. Dongarra, and R. Wolski. The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In *HPDC*, pages 141–152. ACM, 2008.