The Eucalyptus Open-source Cloud-computing System

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

Cloud computing systems fundamentally provide access to large pools of data and computational resources through a variety of interfaces similar in spirit to existing grid and HPC resource management and programming systems. These types of systems offer a new programming target for scalable application developers and have gained popularity over the past few years. However, most cloud computing systems in operation today are proprietary, rely upon infrastructure that is invisible to the research community, or are not explicitly designed to be instrumented and modified by systems researchers.

In this work, we present EUCALYPTUS *– an open-source software framework for cloud computing that implements what is commonly referred to as Infrastructure as a Service (IaaS); systems that give users the ability to run and control entire virtual machine instances deployed across a variety physical resources. We outline the basic principles of the* EUCALYPTUS *design, detail important operational aspects of the system, and discuss architectural trade-offs that we have made in order to allow Eucalyptus to be portable, modular and simple to use on infrastructure commonly found within academic settings. Finally, we provide evidence that* EUCALYPTUS *enables users familiar with existing Grid and HPC systems to explore new cloud computing functionality while maintaining access to existing, familiar application development software and Grid middle-ware.*

1 Introduction

There are many ways in which computational power and data storage facilities are provided to users, ranging from a user accessing a single laptop to the allocation of thousands of compute nodes distributed around the world. Users generally locate resources based on a variety of characteristics, including the hardware architecture, memory and storage capacity, network connectivity and, occasionally, geographic location. Usually this resource location process involves a mix of resource availability, application performance profiling, software service requirements, and administrative connections. While great strides have been made in the HPC and Grid Computing communities [15, 7] toward the creation of resource provisioning standards [14, 18, 33, 38], this process remains somewhat cumbersome for a user with complex resource requirements.

For example, a user that requires a large number of computational resources might have to contact several different resource providers in order to satisfy her requirements. When the pool of resources is finally delivered, it is often heterogeneous, making the task of performance profiling and efficient use of the resources difficult. While some users have the expertise required to exploit resource heterogeneity, many prefer an environment where resource hardware, software stacks, and programming environments are uniform. Such uniformity makes the task of large-scale application development and deployment more accessible.

Recently, a number of systems have arisen that attempt to convert what is essentially a manual large-scale resource provisioning and programming problem into a more abstract notion commonly referred to as elastic, utility, or cloud computing (we use the term "cloud computing" to refer to these systems in the remainder of this work). As the number and scale of cloud-computing systems continues to grow, significant study is required to determine directions we can pursue toward the goal of making future cloud computing platforms successful. Currently, most existing cloud-computing offerings are either proprietary or depend on software that is not amenable to experimentation or instrumentation. Researchers interested in pursuing cloud-computing infrastructure questions have few tools with which to work.

The lack of research tools is unfortunate given that even the most fundamental questions are still unanswered: what is the right distributed architecture for a cloud-computing system? What resource characteristics must VM instance schedulers consider to make most efficient use of the resources? How do we construct VM instance

networks that are flexible, well-performing, and secure? In addition, questions regarding the benefits of cloud computing remain difficult to address. Which application domains can benefit most from cloud computing systems and what interfaces are appropriate? What types of service level agreements should cloud computing provide? How can cloud-computing systems be merged with more common resource provisioning systems already deployed?

Cloud computing systems provide a wide variety of interfaces and abstractions ranging from the ability to dynamically provision entire virtual machines (i.e., Infrastructure-as-a-Service systems such as Amazon EC2 and others [4, 12, 27, 9, 30]) to flexible access to hosted software services (i.e. Software-as-a-Service systems such as salesforce.com and others [37, 20, 21, 29]). All, however, share the notion that delivered resources should be well defined, provide reasonably deterministic performance, and can be allocated and de-allocated on demand. We have focused our efforts on the "lowest" layer of cloud computing systems (IaaS) because here we can provide a solid foundation on top of which language-, service-, and application-level cloud-computing systems can be explored and developed.

In this work, we present EUCALYPTUS: an open-source cloud-computing framework that uses computational and storage infrastructure commonly available to academic research groups to provide a platform that is modular and open to experimental instrumentation and study. With EUCALYPTUS, we intend to address open questions in cloud computing while providing a common open-source framework around which we hope a development community will arise. EUCALYPTUS is composed of several components that interact with one another through well-defined interfaces, inviting researchers to replace our implementations with their own or to modify existing modules. Here, we address several crucial cloud computing questions, including VM instance scheduling, VM and user data storage, cloud computing administrative interfaces, construction of virtual networks, definition and execution of service level agreements (cloud/user and cloud/cloud), and cloud computing user interfaces. In this work, we will discuss each of these topics in more detail and provide a full description of our own initial implementations of solutions within the EUCALYPTUS software framework.

2 Related Work

Machine virtualization projects producing Virtual Machine (VM) hypervisor software [5, 6, 25, 40] have enabled new mechanisms for providing resources to users. In particular, these efforts have influenced hardware design [3, 22, 26] 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 [23, 24, 31, 32, 41] is an active area of research. EUCALYPTUS implements a cloud computing "operating system" that is, by design, hypervisor agnostic. However, the current implementation of the system uses Xen-based virtualization as its initial target hypervisor.

Grid computing must be acknowledged as an intellectual sibling of, if not ancestor to, cloud computing [7, 15, 33, 38]. 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 [16, 17] 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 [34]. Cloud systems often limit the size of an individual request to be tiny fraction of the total available capacity [4] 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 [14]. 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. While EUCALYPTUS is designed to manage and control large collections of distributed resources, it conforms to the design constraints governing cloud systems with regards to federation of administrative authority and resource allocation policies.

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 [4, 1, 8, 10, 11, 19, 37], 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 [30] is a modular open-source virtual machine management framework from academia. Enomalism [12] is an open-source cloud software infrastructure from a start-up company. Virtual Workspaces [27] is a Globus-based [14] system for provisioning workspaces (i.e., VMs), which leverages several pre-existing solutions developed in the grid comput-

ing arena. The Cluster-on-demand [9] project focuses on the provisioning of virtual machines for scientific computing applications. oVirt [35] 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).

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.

3 EUCALYPTUS Design

The architecture of the EUCALYPTUS system is simple, flexible and modular with a hierarchical design reflecting common resource environments found in many academic settings. In essence, the system allows users to start, control, access, and terminate entire virtual machines using an emulation of Amazon EC2's SOAP and "Query" interfaces. That is, users of EUCALYPTUS interact with the system using the exact same tools and interfaces that they use to interact with Amazon EC2. Currently, we support VMs that run atop the Xen [5] hypervisor, but plan to add support for KVM/QEMU [6], VMware [40], and others in the near future.

We have chosen to implement each high-level system component as a stand-alone Web service. This has the following benefits: first, each Web service exposes a well-defined language-agnostic API in the form of a WSDL document containing both operations that the service can perform and input/output data structures. Second, we can leverage existing Web-service features such as WS-Security policies for secure communication between components. There are four high-level components, each with its own Web-service interface, that comprise a EUCALYPTUS installation:

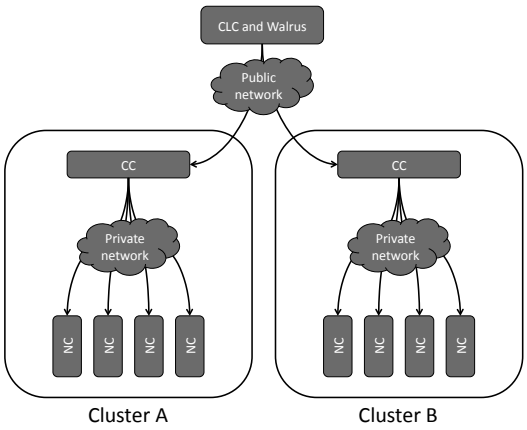

Figure 1. EUCALYPTUS *employs a hierarchical design to reflect underlying resource topologies.*

- **Node Controller** controls the execution, inspection, and terminating of VM instances on the host where it runs.
- **Cluster Controller** gathers information about and schedules VM execution on specific node controllers, as well as manages virtual instance network.
- **Storage Controller (Walrus)** is a put/get storage service that implements Amazon's S3 interface, providing a mechanism for storing and accessing virtual machine images and user data.
- **Cloud Controller** is the entry-point into the cloud for users and administrators. It queries node managers for information about resources, makes high-level scheduling decisions, and implements them by making requests to cluster controllers.

The relationships and deployment locations of each component within a typical small cluster setting are shown in Figure 1.

Node Controller

An Node Controller (NC) executes on every node that is designated for hosting VM instances. An NC queries and controls the system software on its node (i.e., the host operating system and the hypervisor) in response to queries and control requests from its Cluster Controller.

An NC makes queries to discover the node's physical resources – the number of cores, the size of memory, the available disk space – as well as to learn about the state of VM instances on the node (although an NC keeps track of the instances that it controls, instances may be started and stopped through mechanisms beyond NC's control). The information thus collected is propagated up to the Cluster Controller in responses to *describeResource* and *describeInstances* requests.

Cluster Controllers control VM instances on a node by making *runInstance* and *terminateInstance* requests to the node's NC. Upon verifying the authorization – e.g., only the owner of an instance or an administrator is allowed to terminate it – and after confirming resource availability, the NC executes the request with the assistance of the hypervisor. To start an instance, the NC makes a node-local copy of the instance image files (the kernel, the root file system, and the ramdisk image), either from a remote image repository or from the local cache, creates a new endpoint in the virtual network overlay, and instructs the hypervisor to boot the instance. To stop an instance, the NC instructs the hypervisor to terminate the VM, tears down the virtual network endpoint, and cleans up the files associated with the instance (the root file system is not preserved after the instance terminates).

Cluster Controller

The Cluster Controller (CC) generally executes on a cluster front-end machine, or any machine that has network connectivity to both the nodes running NCs and to the machine running the Cloud Controller (CLC). Many of the CC's operations are similar to the NC's operations but are generally plural instead of singular (e.g. *runInstances*, *describeInstances*, *terminateInstances*, *describeResources*). CC has three primary functions: schedule incoming instance run requests to specific NCs, control the instance virtual network overlay, and gather/report information about a set of NCs. When a CC receives a set of instances to run, it contacts each NC component through its *describeResource* operation and sends the runInstances request to the first NC that has enough free resources to host the instance. When a CC receives a *describeResources* request, it also receives a list of resource characteristics (cores, memory, and disk) describing the resource requirements needed by an instance (termed a VM "type"). With this information, the CC calculates how many simultaneous instances of the specific "type" can execute on its collection of NCs and reports that number back to the CLC.

Virtual Network Overlay

Perhaps one of the most interesting challenges in the design of a cloud computing infrastructure is that of VM instance interconnectivity. 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 or with another VM that is co-allocated on the same physical resource. 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. 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.

Within EUCALYPTUS, the CC currently handles the set up and tear down of instance virtual network interfaces in three distinct, administrator defined "modes", corresponding to three common environments we currently support. The first configuration instructs EUCALYPTUS to attach the VM's 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 non-EUCALYPTUS component DHCP requests. The second configuration 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 a free MAC/IP tuple, which is released when the instance is terminated. In these modes, the performance of inter-VM communication is near-native, when VMs are running on the same cluster (any performance overhead is that imposed by the underlying hypervisor implementation), but there is not inter-VM network isolation. Finally, we support a mode in which EUCALYPTUS fully manages and controls the VM networks, providing VM traffic isolation, the definition of ingress rules (configurable firewalls) between logical sets of VMs, and the dynamic assignment of public IP addresses to VMs at boot and run-time.

In this mode, the users are allowed to attach VMs, at boot time, to a "network" that is named by the user. Each such network is assigned a unique VLAN tag by EUCALYPTUS, depicted in Figure 2, as well as a unique IP subnet from a range specified by the administrator of EUCALYPTUS in advance (typically a private IP range). In this way, each set of VMs within a given named network is isolated from VMs on a different named network at using VLANs, and further using IP subnetting. The CC acts as a router between VM subnets, with the default policy blocking all IP traffic between VM networks. If the user wishes, they may associate ingress rules with their named networks, allowing for example ICMP ping traffic to and from the public Internet, but only allowing SSH connections between VMs that they control. The CC uses the Linux iptables packet filtering system to implement and

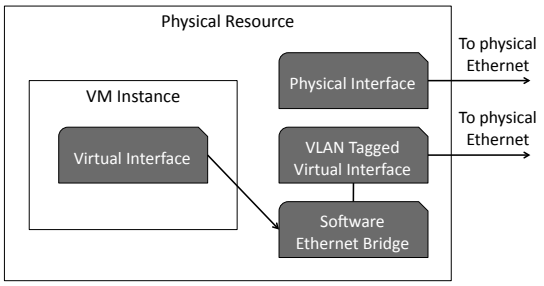

Figure 2. *Each* EUCALYPTUS *VM instance is assigned a virtual interface that is connected to a software Ethernet bridge on the physical machine, to which a VLAN tagged interface is further connected.*

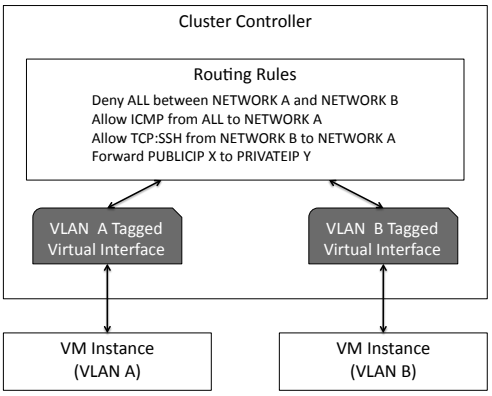

Figure 3. *The CC uses the Linux iptables packet filtering system to allow users to define inter-VM network ingress rules, and to assign public IP addresses dynamically at boot or run-time.*

control VM network ingress rules. Finally, note that all VMs in this mode are typically assigned form a pool of private IP addresses, and as such cannot be contacted by external systems. To manage this situation, the CC allows the administrator to specify a list of public IPv4 addresses that are unused, and provides the ability for users to dynamically request that an IP from this collection be assigned to a VM at boot or run-time; again using the Linux iptables Network Address Translation (NAT) facilities to define dynamic Destination NAT (DNAT) and Source NAT (SNAT) rules for public IP to private IP address translation (see Figure 3 for details).

From a performance perspective, the solution we employ exhibits near native speed when two VMs within a given named network within a single cluster communicate with one another. When VMs on different named networks need to communicate, our solution imposes one extra hop through the CC machine which acts as an IP router. Thus, we afford the user with the ability to choose, based on their specific application requirements, between native performance without inter-VM communication restrictions, or suffer an extra hop but gain the ability to restrict inter-VM communication.

When VMs are distributed across clusters, we provide a manual mechanism for linking the cluster front-ends via a tunnel (for example, VTUN [28]). Here, all VLAN tagged Ethernet packets from one cluster are tunneled to another over a TCP or UDP connection. Performance of cross-cluster VM communication is likely to be dictated, primarily, by the speed of the wide area link. However, the performance impact of this tunnel can be substantial if the link between clusters is sufficiently high performance. More substantial performance evaluation study of this and other cloud networking systems is being performed, but is beyond the scope of this work.

Storage Controller (Walrus)

EUCALYPTUS includes Walrus, a data storage service that leverages standard web services technologies (Axis2, Mule) and is interface compatible with Amazon's Simple Storage Service (S3) [36]. Walrus implements the REST (via HTTP), sometimes termed the "Query" interface, as well as the SOAP interfaces that are compatible with S3. Walrus provides two types of functionality.

- Users that have access to EUCALYPTUS can use Walrus to stream data into/out of the cloud as well as from instances that they have started on nodes.
- In addition, Walrus acts as a storage service for VM images. Root filesystem as well as kernel and ramdisk images used to instantiate VMs on nodes can be uploaded to Walrus and accessed from nodes.

Users use standard S3 tools (either third party or those provided by Amazon) to stream data into and out of Walrus. The system shares user credentials with the Cloud Controller's canonical user database.

Like S3, Walrus supports concurrent and serial data transfers. To aid scalability, Walrus does not provide locking for object writes. However, as is the case with S3, users are guaranteed that a consistent copy of the object will be saved if there are concurrent writes to the same object. If a write to an object is encountered while there is a previous write to the same object in progress, the previous write is invalidated. Walrus responds with the MD5 checksum of the object that was stored. Once a request has been verified, the user has been authenticated as a valid EUCALYPTUS user and checked against access control lists for the object that has been requested, writes and reads are streamed over HTTP.

Walrus also acts as an VM image storage and management service. VM root filesystem, kernel and ramdisk im-

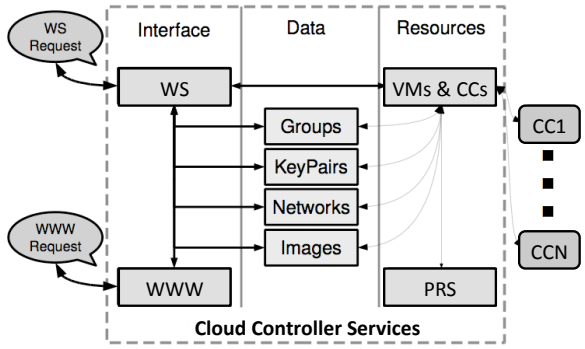

Figure 4. *Overview of Cloud Controller services. Dark lines indicate the flow of user requests while light lines correspond to inter-service system messages.*

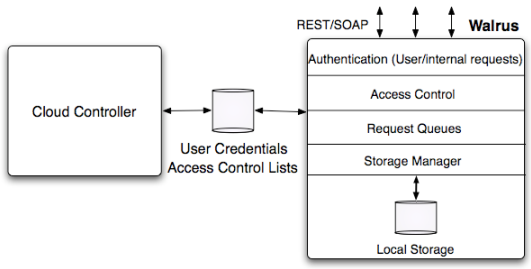

Figure 5. EUCALYPTUS *includes Walrus, a S3 compatible storage management service for storing and accessing user data as well as images.*

ages are packaged and uploaded using standard EC2 tools provided by Amazon. These tools compress images, encrypt them using user credentials, and split them into multiple parts that are described in a image description file (called the *manifest* in EC2 parlance). Walrus is entrusted with the task of verifying and decrypting images that have been uploaded by users. When a node controller (NC) requests an image from Walrus before instantiating it on a node, it sends an image download request that is authenticated using an internal set of credentials. Then, images are verified and decrypted, and finally transferred. As a performance optimization, and because VM images are often quite large, Walrus maintains a cache of images that have already been decrypted. Cache invalidation is done when an image manifest is overwritten, or periodically using a simple least recently used scheme.

Walrus is designed to be modular such that the authentication, streaming and back-end storage subsystems can be customized by researchers to fit their needs.

Cloud Controller

The underlying virtualized resources that comprise a EUCALYPTUS cloud are exposed and managed by, the Cloud Controller (CLC). The CLC is a collection of web-services which are best grouped by their roles into three categories:

- *Resource Services* perform system-wide arbitration of resource allocations, let users manipulate properties of the virtual machines and networks, and monitor both system components and virtual resources.
- *Data Services* govern persistent user and system data and provide for a configurable user environment for formulating resource allocation request properties.
- *Interface Services* present user-visible interfaces, handling authentication & protocol translation, and expose system management tools providing.

The Resource services process user virtual machine control requests and interact with the CCs to effect the allocation and deallocation of physical resources. A simple representation of the system's resource state (SRS) is maintained through communication with the CCs (as intermediates for interrogating the state of the NCs) and used in evaluating the realizability of user requests (vis a vis service-level agreements, or SLAs). The role of the SRS is executed in two stages: when user requests arrive, the information in the SRS is relied upon to make an admission control decision with respect to a user-specified service level expectation. VM creation, then, consists of *reservation* of the resources in the SRS, downstream request for VM creation, followed by *commitment* of the resources in the SRS on success, or *rollback* in case of errors.

The SRS then tracks the state of resource allocations and is the source of authority of changes to the properties of running reservations. SRS information is leveraged by a production rule system allowing for the formulation of an event-based SLA scheme. Application of an SLA is triggered by a corresponding event (e.g., network property changes, expiry of a timer) and can evaluate and modify the request (e.g., reject the request if it is unsatisfiable) or enact changes to the system state (e.g., time-limited allocations). While the system's representation in the SRS may not always reflect the actual resources, notably, the likelihood and nature of the inaccuracies can be quantified and considered when formulating and applying SLAs. Further, the admission control and the runtime SLA metrics work in conjunction to: ensure resources are not overcommitted and maintain a conservative view on resource availability to mitigate possibility of (service- level) failures.

A concrete example from our implementation allows users to control the cluster to be used for the VM allocations by specifying the "zone" (as termed by Ama-

zon). Further, we have extended the notion of zone to *meta-zones* which advertise abstract allocation policies. For example, the "any" meta-zone will allocate the user-specified number of VMs to the emptiest cluster, but, in the face of resource shortages, overflow the allocation to multiple clusters.

The middle tier of Data Services handle the creation, modification, interrogation, and storage of stateful system and user data. Users can query these services to discover available resource information (images and clusters) and manipulate abstract parameters (keypairs, security groups, and network definitions) applicable to virtual machine and network allocations. The Resource Services interact with the Data Services to resolve references to user provided parameters (e.g., keys associated with a VM instance to be created). However, these services are not static configuration parameters. For example, a user is able to change, what amounts to, firewall rules which affect the ingress of traffic. The changes can be made offline and provided as inputs to a resource allocation request, but, additionally, they can be manipulated while the allocation is running. As a result, the services which manage the networking and security group data persistence must also act as agents of change on behalf of a user request to modify the state of a running collection of virtual machines and their supporting virtual network.

In addition to the programmatic interfaces (SOAP and "Query"), the Interface tier also offers a Web interface for cloud users and administrators. Using a Web browser, users can sign up for cloud access, download the cryptographic credentials needed for the programmatic interface, and query the system, e.g., about available disk images. The administrators can, additionally, manage user accounts, inspect the availability of system components.

Lastly, the collection of interface web services advertises entry points for user requests using a variety of interface specifications (e.g., EC2's SOAP & "Query", S3's SOAP & REST') where single sign authentication is done according to the best practices common among cloud vendors. Users can make requests using either the EC2 SOAP or EC2 "Query" protocols [4]. In particular this has allowed the wide variety of tools which comply with the EC2 and S3 interfaces to work without modification. The key design goal achieved by the interface services is a insulation of the internal communication data types by mapping requests from these disparate protocols to an independent system-internal protocol. Consequently, internal services are unconcerned with details of the outward-facing interfaces utilized by users while being able to mimic the functionality, syntax, and structure of the interface primitives preserving existing investment in tools and code.

4 Conclusions

The EUCALYPTUS system is built to allow administrators and researchers the ability to deploy an infrastructure for user-controlled virtual machine creation and control atop existing resources. Its hierarchical design targets resource commonly found within academic and laboratory settings, including but not limited to small- and medium-sized Linux clusters, workstation pools, and server farms. We use a virtual networking solution that provides VM isolation, high performance, and a view of the network that is simple and flat. The system is highly modular, with each module represented by a well-defined API, enabling researchers to replace components for experimentation with new cloud-computing solutions. Finally, the system exposes its feature set through a common user interface currently defined by Amazon EC2 and S3. This allows users who are familiar with EC2 and S3 to transition seamlessly to a EUCALYPTUS installation by, in most cases, a simple addition of a command-line argument or environment variable, instructing the client application where to send its messages.

In sum, this work aims to illustrate the fact that the EUCALYPTUS system has filled an important niche in the cloud-computing design space by providing a system that is easy to deploy atop existing resources, that lends itself to experimentation by being modular and open-source, and that provides powerful features out-of-the-box through an interface compatible with Amazon EC2.

Presently, we and our users have successfully deployed the complete system on resources ranging from a single laptop (EC2 on a laptop) to small Linux clusters (48 to 64 nodes). The system is being used to experiment with HPC and cloud computing by trying to combine cloud computing systems like EUCALYPTUS and EC2 with the Teragrid (presented as a demo at SuperComputing'08 as part of the VGrADS [39] project), as a platform to compare cloud computing systems' performance, and by many users who are interested in experimenting with a cloud computing system on their own resources.

In addition, we have made a EUCALYPTUS installation available to all who wish to try out the system without installing any software [13]. Our experience so far has been extremely positive, leading us to the conclusion that EUCALYPTUS is helping to provide the research community with a much needed, open-source software framework around which a user-base of cloud-computing researchers can be developed.

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 Web Services home page. http://aws.amazon.com/.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] Eucalyptus Public Cloud (EPC). http://eucalyptus.cs.ucsb.edu/wiki/EucalyptusPublicCloud/.

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

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

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

[17] 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.

[18] D. Gannon. Programming the grid: Distributed software components, 2002.

[19] Google – http://www.google.com/.

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

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

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

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

[24] 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.

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

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

[27] 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.

[28] M. Krasnyansky. VTun-Virtual Tunnels over TCP/IP networks, 2003.

[29] 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.

[30] 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.

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

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

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

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

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

[36] Amazon simple storage service api (2006-03-01) – http://docs.amazonwebservices.com/AmazonS3/2006-03-01/.

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

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

[39] Virtual Grid Application Development Software project. http://vgrads.rice.edu/.

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

[41] 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.