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 amounts of data and computational resources through a variety of interfaces. Many extant systems have in common the notion that resources can be acquired and released on-demand and that the user interface be kept fairly simple. In addition, resources provided by cloud computing systems hide a great deal of information from the user through virtualization (physical location of the resource, precise architectural details of the compute resources). 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 interested in cloud computing systems.

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, 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.*

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 [10, 4] toward the creation of resource provisioning standards [9, 11, 17, 19], 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 [6, 7, 14, 5, 16]) to flexible access to hosted software services (i.e. Software-as-a-Service systems such as salesforce.com and others [18, 12, 13, 15]). 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, 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 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

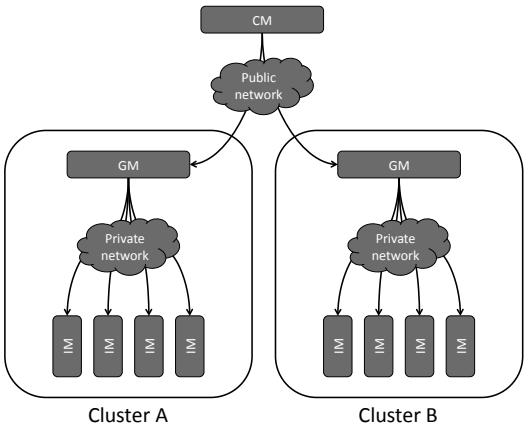

Figure 1. EUCALYPTUS *employs a hierarchical design to reflect underlying resource topologies.*

that run atop the Xen [2] hypervisor, but plan to add support for KVM/QEMU [3], VMware [21], 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 three high-level components, each with its own Web-service interface, that comprise a EUCALYPTUS installation:

- **Instance Manager** controls the execution, inspection, and terminating of VM instances on the host where it runs.
- **Group Manager** gathers information about and schedules VM execution on specific instance managers, as well as manages virtual instance network.
- **Cloud Manager** 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 group managers.

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

Instance Manager

An Instance Manager (IM) executes on every node that is designated for hosting VM instances. An IM queries and controls the system software on its node (i.e., the

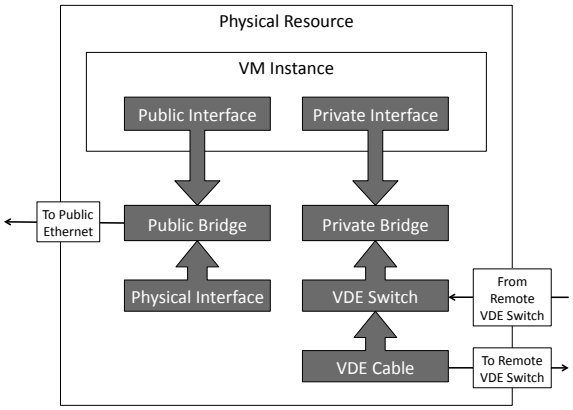

Figure 2. *Each instance is assigned a public interface connected to the physical Ethernet, and a private interface connected to a VDE virtual Ethernet.*

host operating system and the hypervisor) in response to queries and control requests from its Group Manager.

An IM 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 IM keeps track of the instances that it controls, instances may be started and stopped through mechanisms beyond IM's control). The information thus collected is propagated up to the Group Manager in responses to *describeResource* and *describeInstances* requests.

Group Managers control VM instances on a node by making *runInstance* and *terminateInstance* requests to the node's IM. 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 IM executes the request with the assistance of the hypervisor. To start an instance, the IM 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 IM 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).

Group Manager

The Group Manager (GM) generally executes on a cluster front-end machine, or any machine that has network connectivity to both the nodes running IMs and to the machine running the Cloud Manager (CM). Many of the GM's operations are similar to the IM's operations but are generally plural instead of singular (e.g. *runInstances*, *describeInstances*, *terminateInstances*, *describeResources*).

GM has three primary functions: schedule incoming instance run requests to specific IMs, control the instance virtual network overlay, and gather/report information about a set of IMs. When a GM receives a set of instances to run, it contacts each IM component through its *describeResource* operation and sends the runInstances request to the first IM that has enough free resources to host the instance. When a GM 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 GM calculates how many simultaneous instances of the specific "type" can execute on its collection of IMs and reports that number back to the CM.

Finally, the GM is responsible for setting up and controlling the instance virtual network over which all VM instances within a user's set of instances may communicate, even when those instances may be running on physical machines distributed over wide areas and shielded by firewalls. To implement such a network overlay, the GM uses software from the Virtual Distributed Ethernet (VDE) project [20]. This software implements the Ethernet protocol in software, providing virtual Ethernet "switches" and "cables" to be run as user-space processes. Each component (IM, GM, CM) in EUCALYPTUS runs a single VDE switch, and VDE cables (encrypted UDP connections) are established between as many switches as possible. As long as there is at least one cable to each switch, the VDE network provides a fully connected Ethernet network to which instance's private network interfaces are attached.

Once the GM has set up the virtual Ethernet overlay, each instance is given both a "public" and "private" interface, which are connected via software Ethernet bridges to the local Ethernet and VDE overlay, respectively. In Figure 2, we show how each instance is logically connected to both types of network. This configuration allows instances within a cluster to communicate over the "fast" local network, and also to use the empirically slower virtual network to communicate with instances physically residing in other clusters. To the owner of the instances, the overlay provides the appearance of a flat subnet to which all instances are connected.

Cloud Manager

The underlying resources that comprise a EUCALYPTUS cloud are exposed to users, and managed by, the Cloud Manager (CM). The CM, like the system overall, is a three-tiered design as depicted in Figure 3. The tiers are distinct in their roles and concomitant data statefulness/consistency requirements:

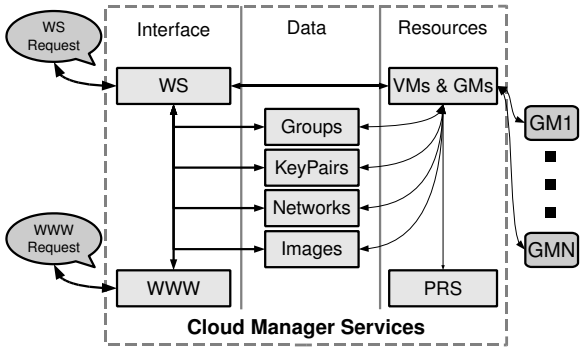

Figure 3. *Overview of Cloud Manager services. Dark lines indicate the flow of user requests while light lines correspond to inter-service system messages.*

- *Interface Services* present user-visible interfaces, handling authentication & protocol translation, and expose system management tools.
- *Data Services* govern persistent user and system data.
- *Resource Services* arbitrate allocation and monitoring of resources and active VM allocations.

Our implementation supports extension and modification at granularities ranging from complete service replacement to fine-grained tuning (e.g., of user interfaces or allocation policy) through well-defined message and language interfaces, respectively. System-enforced separation between interface and internal message protocol insulates service implementations from user protocol details. Similarly, configurable service-ensemble organization decouples individual service implementations from runtime coordination dependencies.

The Interface's WS service advertises a single multi-protocol endpoint for authenticating and consuming user requests while also translating the request to an internal protocol. Users can make requests using either the EC2 SOAP or EC2 "Query" protocols [1] (which, additionally, require incompatible authentication mechanisms: X509 and HMAC signatures, respectively). This duality has been achieved through the introduction and utilization of pluggable request handling interfaces in the supporting Web services stack software. The key function of the Interface service is the mapping of 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 benefitting from enforcement of message validation requirements.

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: approve, disable, and delete them. Currently, images can be added to the system by the administrator only with a command-line tool, but we expect the Web interface to support complete administrative functionality in the future.

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. Conversely, the VM service resolves references to these resources when realizing user requests.

The Resource services process user virtual machine control requests and interact with the GMs to effect the allocation and deallocation of resources. A simple representation of the system's physical resource state (PRS) is maintained through communication with the GMs and used in evaluating the realizability of user requests (vis a vis service-level agreements, or SLAs). VM control requests are handled asynchronously with respect to the user and, therefore, transactions delimit changes to the PRS. For example, VM creation consists of *reservation* of the resources in the PRS, downstream request for VM creation, followed by *commitment* of the resources in the PRS on success, or *rollback* in case of errors.

PRS information is then exploited by an event-based SLA scheme to evaluate the satisfiability of user requests and enforce system policy. Application of an SLA is triggered by a corresponding event (e.g., VM allocation request, 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 PRS 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.

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 Amazon). 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.

3 Discussion

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 architectures 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 software that provides a virtual Ethernet overlay to connect VM instances that execute in isolated networks, providing users 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. This allows users who are familiar with EC2 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.

4 Conclusion

In this work, we present EUCALYPTUS: an open-source implementation of an IaaS system. 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, 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 made have a EUCALYPTUS installation available to all who wish to try out the system without installing any software [8]. 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] Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.

[2] 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.

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

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

[5] 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.

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

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

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

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

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

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

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

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

[14] 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.

[15] 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.

[16] 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.

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

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

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

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

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