5-2010

Practical Implementation of the Virtual Organization Cluster Model

Michael Fenn
Clemson University, michaelfenn87@gmail.com

Practical Implementation of the Virtual Organization Cluster Model

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

---

by
Michael Fenn
May 2010

Accepted by:
Dr. Sebastien Goasguen, Committee Chair
Dr. Mike Westall
Dr. Walt Ligon

Abstract

Virtualization has great potential in the realm of scientific computing because of its inherent advantages with regard to environment customization and isolation. Virtualization technology is not without it's downsides, most notably, increased computational overhead. This thesis introduces the operating mechanisms of grid technologies in general, and the Open Science Grid in particular, including a discussion of general organization and specific software implementation. A model for utilization of virtualization resources with separate administrative domains for the virtual machines (VMs) and the physical resources is then presented. Two well-known virtual machine monitors, Xen and the Kernel-based Virtual Machine (KVM), are introduced and a performance analysis conducted. The High-Performance Computing Challenge (HPCC) benchmark suite is used in conjunction with independent High-Performance Linpack (HPL) trials in order to analyze specific performance issues. Xen was found to introduce much lower performance overhead than KVM, however, KVM retains advantages with regard to ease of deployment, both of the VMM itself and of the VM images. KVM's snapshot mode is of special interest, as it allows multiple VMs to be instantiated from a single image located on a network store.

With virtualization overhead shown to be acceptable for high-throughput computing tasks, the Virtual Organization Cluster (VOC) Model was implemented as a prototype. Dynamic scaling and multi-site scheduling extensions were also successfully implemented using this prototype. It is also shown that traditional overlay networks have scaling issues and that a new approach to wide-area scheduling is needed.

The use of XMPP messaging and the Google App Engine service to implement a virtual machine monitoring system is presented. Detailed discussions of the relevant sections of the XMPP protocol and libraries are presented. XMPP is found to be a good choice for sending status information due to its inherent advantages in a bandwidth-limited NAT environment.

Thus, it is concluded that the VOC Model is a practical way to implement virtualization of high-throughput computing tasks. Smaller VOCs may take advantage of traditional overlay networks whereas larger VOCs need an alternative approach to scheduling.

Acknowledgments

I would like to thank Dr. Sebastien Goasguen, who recognized a little spark of potential in one of his Distributed Systems students. He's consistently pushed me toward greater and greater challenges, and without his guidance I would have never have participated in the Google Summer of Code or even considered graduate school. His guidance has helped me bring out the best in myself.

I would also like to thank Mike Murphy, who was brave enough to give a lowly undergrad root access to his cluster. Though we have had our share of technical "discussions," I am deeply indebted to him for passing on his truly vast knowledge of Linux.

I would like to acknowledge Jerome Lauret of Brookhaven National Laboratory for his help in conducting tests with the STAR VO and David Wolinsky of the University of Florida for his help in conducting IPOP scalability testing.

Many thanks are also due to the whole Cyberinfrastructure Research Group: Dru, Brandon, Jordan, Josh, Ben, Kristen, Linton, and Lance.

Table of Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Virtual Organizations (VOs) allow collaboration among scientists and utilization of diverse, geographically distributed computing resources. These VOs often have dynamic changes in their membership and requirements, especially in terms of their computing needs over time [1]. Given the diverse nature of VOs, as well as the challenges involved in providing suitable computing environments to each VO, Virtual Machines (VMs) are a promising abstraction mechanism for providing grid computing services [2]. Cluster computing systems, including services and middleware that can take advantage of several available virtual machine monitors (VMMs) [3], have already been constructed inside VMs [4, 5, 6, 7]. A cohesive view of virtual clusters for grid-based VOs was presented by Murphy, Fenn, and Goasguen [8], and this thesis will describe the model in some detail.

Implementing computational clusters with traditional multiprogramming systems may result in complex systems that require different software sets for different users. Each user is also limited to the software selection chosen by a single system administrative entity, which may be different from the organization sponsoring the user. Virtualization provides a mechanism by which each user entity might be given its own computational environment. Such virtualized environments would permit greater end-user customization at the expense of some computational overhead [2].

While traditional grid computing research has made significant progress on protocols and standards for sharing resources among VOs, individual clusters must still balance the needs of each supported VO alongside the needs of its local user base. As a result, operating and managing cluster computing resources has been more complex for system administrators trying to support multiple VOs on the same cluster. Currently, users must make accommodations due to the shared nature

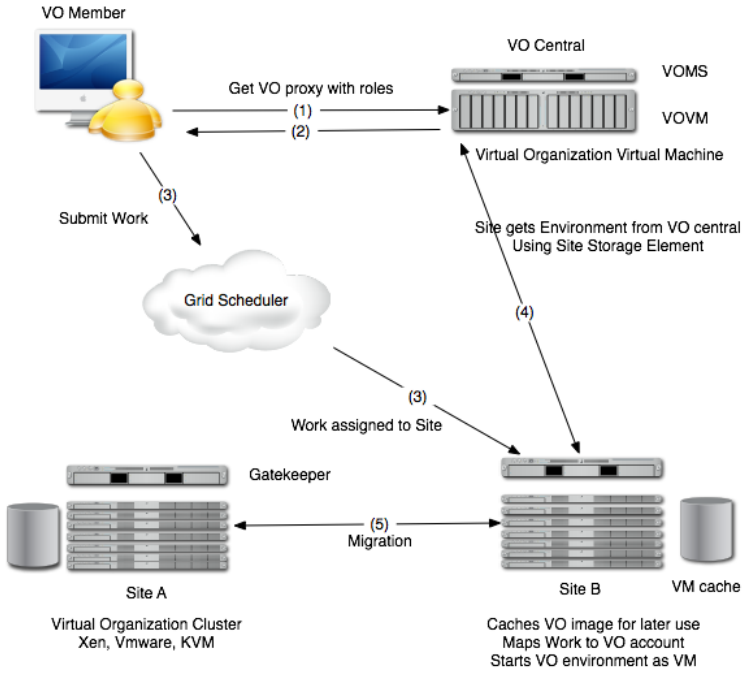

Figure 1.1: Grid-Based Use Case for a VOC

of their resources. For example, if a particular user needs a software package that conflicts with a package needed by another user, the systems administrator must choose between the needs of the two users or must implement a complex workaround. Virtualization can be used to provide a homogeneous computing environment to the users while spanning geographically dispersed, heterogeneous resources connected via grid protocols. Therefore, it is important to focus on the setup, operation and performance of the physical systems that support virtual clusters dedicated to individual VOs. [9, 8]

The primary motivation for the work described here is to enable a scalable, easy-to-maintain system on which each Virtual Organization can deploy its own customized environment. Such environments will be scheduled to execute on the physical fabric, thereby permitting each VO to schedule jobs on its own private cluster. Additionally, due to the ease with which VMs can be instantiated, this environment should be able to scale dynamically as well as span multiple geographical sites.

Figure 1.1, first presented by Murphy, Fenn, and Goasguen [8], represents an idealized use case of Virtual Organization Clusters (VOCs) in a manner based on the operating principles of the Open Science Grid [10] – a grid infrastructure known to support VOs instead of individual users. In the figure, "VO Central" is a database run by the VO manager. It contains a list of members and

their associated privileges stored in the Virtual Organization Manager Service (VOMS), and a set of computing environments (in the form of virtual machine images) stored in the "Virtual Organization Virtual Machine (VOVM)," also known to as a VOC node. When a VO member wants to send work to the grid, a security proxy is obtained from her VOMS server, and the work is submitted to a VO meta-scheduler (casually depicted as a cloud in this figure). Once work is assigned to a site, this site downloads the proper VM either from the VOC node or from its own VM cache. These data transfers can be done through the OSG data-transfer mechanisms (i.e. dCache [11] with SRM) and can use the GridFTP protocol. If a site becomes full, work can be migrated to another site using VM migration mechanisms. This use case represents an ideal form of grid operation, which would provide a homogeneous computing environment to the users. Each VO would maintain its own environment and update its own software packages. Each physical site could determine its own local policies and operating system setup. For this use case to become reality, this thesis presents the Virtual Organization Cluster (VOC) model, which expands on previous knowledge of virtual machines, cluster setup, and grid computing.

Central to the VOC model is the Virtual Organization Cluster (VOC). A VOC is a cluster made of virtual machines configured to support a single VO and deployed by a Virtualization Service Provider (VSP). A VSP and a VOC have been developed at the Cyberinfrastructure Research Laboratory at Clemson University [12], where they appear as a resource on the Open Science Grid. This VOC is deployed on physical cluster with both the Kernel-based Virtual Machine (KVM) virtual machine monitor and Xen hypervisor. The VOC is composed of CentOS 5 VOC nodes, providing VO compute services to the Engage OSG VO. Initial benchmarking results indicate that the VOC is suitable for High Throughput Computing (HTC) (e.g. vanilla-universe Condor [13] jobs). Severe networking overhead is present in KVM, creating large penalties in jobs which heavily leverage the network, including those using the Message Passing Interface (MPI), such as a subset of the High Performance Computing Challenge (HPCC) benchmarking suite. [9]

Dynamic scaling (also known as dynamic provisioning) is the process by which a virtual environment can be sized according to load. Given that the benchmarking results showed acceptable overheads for HTC jobs, the prototype VOC has been extended to support dynamic scaling. This VOC is controlled by a *watchdog* process which expands and shrinks the size of the VOC in response to the size of the Condor job queue. This implementation successfully achieves the goal of providing a dynamically scalable VOC with predictable behaviors.

A further extension of this line of research is presented that allows a VOC to span multiple physical sites whose networking topology includes various Network Address Translation (NAT) boundaries. This is accomplished by adding a previously developed overlay network tool, known as Internet Protocol Over Peer-to-peer (IPOP) [14]. IPOP creates a virtual network overlay which can span NAT boundaries and thus make various hosts appear to be on the same subnet. Concerns about the scalability of IPOP were presented, and thus scaling tests were performed. Empirical tests show that IPOP has a scaling limit of approximately 500 nodes. This is an obstacle to the large-scale deployment of VOCs, but small and medium-scale deployments remain viable. One such test is presented, with VMs from Amazon's Elastic Compute Cloud (EC2) service joining the scheduling pool provided by the local prototype VOC.

Due to the scaling limits discovered in IPOP, an alternative strategy for scheduling HTC jobs across NAT boundaries is presented. An implementation of a monitoring front-end has been deployed to Google App Engine that provides user access to a deployment of OpenNEbula (ONE) [15]. ONE is a virtual infrastructure manager that provides for the dynamic placement of virtual machines onto virtualization resources. The VMs started by ONE have an Extensible Messaging and Presence Protocol (XMPP) monitoring program running at regular intervals which reports status information back to the front-end. The user may interact with the front-end via a Web browser and HTTP or via an XMPP chat client. See Figure 1.2 for the general architecture of the implemented system.

The remainder of this work is organized into several chapters covering background material regarding related works (Chapter 2), the organization of the Open Science Grid (Chapter 3), the software stack used in the Open Science Grid (Chapter 4), and the Virtual Organization Cluster Model (Chapter 5). The effort of creating a prototype implementation of a VOC is described in Chapter 6, whereas the implementation of a cloud-based monitoring solution is described in Chapter 7. Experimental results are detailed in Chapter 8 and final conclusions are drawn in Chapter 9.

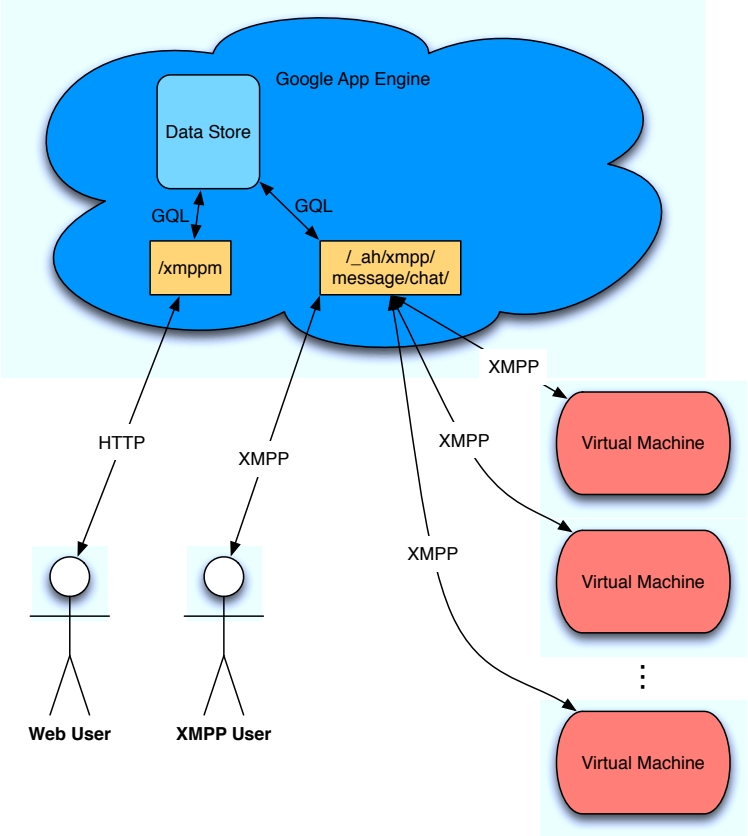

Figure 1.2: Architecture of the monitoring system

Chapter 2

Related Work

Operating system virtualization has been proposed as a mechanism for offering different environments to different users sharing a single physical infrastructure. Figueiredo *et. al.* proposed the use of virtualization systems to support multiplexing among different users, where each user entity could have administrative access to its own virtual environment. Grid computing systems designed in this way could be site-independent, permitting virtual clusters to be executed on different physical systems owned by different entities. Furthermore, users could be better isolated from each other using virtualization systems, compared to shared multiprogramming systems. [2, 9]

Any work relating to virtual clusters must address issues regarding the provisioning and deployment of virtual machines. Middleware designed to facilitate the deployment of clusters of virtual machines exists. Several examples of these middleware-oriented projects are Globus Workspaces [16, 17], VMPlants [5], DVC [18], virtual disk caching [19], and In-VIGO [4, 20, 6, 7].

All virtual clusters must be implemented on top of a physical cluster that provides virtualization services. Several software packages with the purpose of allowing rapid deployment of physical clusters exist: including OSCAR [21], Rocks [22, 23, 24], and Cluster-On-Demand (COD) [25]. In particular, Rocks provides a mechanism for easy additions of software application groups via *rolls*, or meta-packages of related programs and libraries [26]. The OSCAR meta-package system also permits related groups of packages to be installed onto a physical cluster system [21]. These packages serve as the basis of the physical support model for VOCs.

A variety of networking libraries have been developed which display promise for use with multi-site VOCs. Both Virtual Distributed Ethernet (VDE) [27] and Virtuoso [28] provide low-

level virtualized networks that can be utilized for interconnecting VMs. Furthermore, wide-area connectivity of VMs can be achieved through the use of tools such as Wide-area Overlays of virtual Workstations (WOW) [29] and Violin [30]. OpenVPN is an open-source virtual private network (VPN) solution that facilitates the creation of point-to-point or one-to-many tunnels between hosts. It satisfies the requirements of the VOC model, but is lacking in the ability to autonomically adjust to the addition or removal of clients [31]. Internet Protocol Over Peer-to-peer (IPOP) uses a peer-to-peer architecture to create an overlay network [14]. IPOP Brunet is a software library written in C# and Mono that allows interaction with IPOP [32]. IPOP is self-configuring and allow nodes behind various Network Address Translation (NAT) gateways to appear to be on the same private subnet.

Unlike prior cluster computing and virtualization research, the cluster virtualization model described in this thesis focuses on customizing environments for individual VOs instead of individual physical sites. Since *a priori* knowledge of a particular VO's scientific computing requirements is not always available, this model makes few assumptions about the operating environment desired by each individual VO. As a result, the focus of the physical system configuration is to support VMs with minimal overhead and maximal ease of administration. Moreover, the system should be capable of supporting both high-throughput and high-performance distributed computing needs on a per-VO basis, imposing network performance requirements to support MPI and similar packages.[9]

The Extensible Messaging and Presence Protocol (XMPP) is an open protocol aimed at providing real time push notifications and subscription information. The protocol was originally known as Jabber and its purpose was to provide an open instant messaging protocol. The instant messaging use is still XMPP's most well known application. The core XMPP protocol is defined in RFC 3920 while the instant messaging and presence components are defined in RFC 3921. Google App Engine (GAE) [33] is a service provided by Google that allows users to run web applications on Google's own infrastructure. GAE provides both Python and Java runtime environments, as well as a web application framework to assist in this task.

Saint-Andre and Meijer [34] discuss XMPP from an architectural point of view. They describe how XMPP is a form of streaming XML with the core unit being the stanza instead of the document as with traditional uses of XML. They also describe the use of DNS SRV records for the look-up of XMPP servers. Stout et. al. [35] provide a discussion of the Kestrel software package. Kestrel performs many of the monitoring functions which are also present in the implementation

described in this paper but lacks a cloud-based platform from which to provide its service. A user wishing to deploy Kestrel must maintain their own machine on which to run the XMPP server. XMPP has been used in the realm of bioinformatics as a replacement for HTTP-based web services [36]. Christensen provides a description of a Transaction Processing Monitor implemented with XMPP [37]. Bernstein et. al. have proposed a plan for interoperability of cloud services with XMPP root servers as a component [38].

Google App Engine has been described in several survey papers that cover the various cloud computing providers, but so far has not been critically studied in an academic environment [39, 40].

Chapter 3

Organization of the Open Science Grid

Grid computing is a type of distributed computing that seeks to solve the "grid problem" defined by Foster as "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources—what we refer to as virtual organizations [1]." There are several real-world grids including (but not limited to) TeraGrid, the Open Science Grid (OSG), and The Enabling Grids for E-sciencE (EGEE). This thesis will focus on OSG. OSG is comprised of two main types of entities: sites which provide compute resources and Virtual Organizations (VOs) which are composed of like-minded users. Four essential facets of OSG are its public key infrastructure (Section 3.1), VOs (Section 3.2), sites (Section 3.3), and trust model (Section 3.4). Some material in this chapter has been previously presented to the Calhoun Honors College of Clemson University as part of the author's undergraduate honors thesis.[9]

3.1 Public Key Infrastructure

Every large-scale distributed system needs some method of user authentication. OSG uses a Public Key Infrastructure (PKI) that allows sites to authenticate users in a distributed manner. Key components of this (and any) PKI are public key cryptography (Section 3.1.1), certificate authorities (Section 3.1.2), and certificate revocation mechanisms (Section 3.1.3). [9]

3.1.1 Public Key Cryptography

If a Public Key Infrastructure (PKI) is to provide confidentiality to its users, it must provide each entity with two types of cryptographic keys: a public key and a private key. The public key is publicly available and is generally embedded into a certificate. This certificate contains the cryptographic key along with some identifying information. The private key contains another cryptographic hash and should be kept private. The encryption algorithms used in this public-key cryptography work in such a way that any data encrypted with the a public key can only be decrypted with the corresponding private key, and any data encrypted with the private key can only be decrypted with the public key. Therefore, as long as each entity has both a public key and a private key, messages can be exchanged without having previously shared a secret cipher. For example, as illustrated in Figure 3.1, suppose Alice wishes to send a message to Bob. Alice would first obtain Bob's public key by some method, either from Bob directly or through a centralized database. Alice would then encrypt her message with Bob's public key and transmit it to him. Bob could then decrypt Alice's message with his private key. Thus, at no time has Bob had to send his private key across the ether. Bob can then perform the same procedure in reverse to send his reply to Alice, again without needing to know any kind of shared secret. [9]

3.1.2 Certificate Authorities

Authentication in OSG uses this basic mechanism with a few added details. It is not enough to be able to confidentially exchanges messages. A user also needs to be confident of the identity of the receiving part, *i. e.* he needs to be confident in the integrity of the communication. Integrity can be assured by the inclusion of an entity known as a Certificate Authority (CA). In the above example, while Alice can be assured that the message she sent with a given public key can only be decrypted by a person with a given private key, she has no way of verifying Bob's identity. Bob will receive her message, but she has no idea whether or not Bob is a legitimate user. This is where the certificate authority comes into play. Bob's public key is included as a part of Bob's certificate. Bob's certificate was created by taking Bob's public key and adding some identifying information, including the CA's digital signature. The CA creates this signature by taking Bob's unsigned certificate, encrypting it with the CA's private key, and appending this signature to the original certificate. Anyone can now decrypt the signature with the CA's public key and verify that

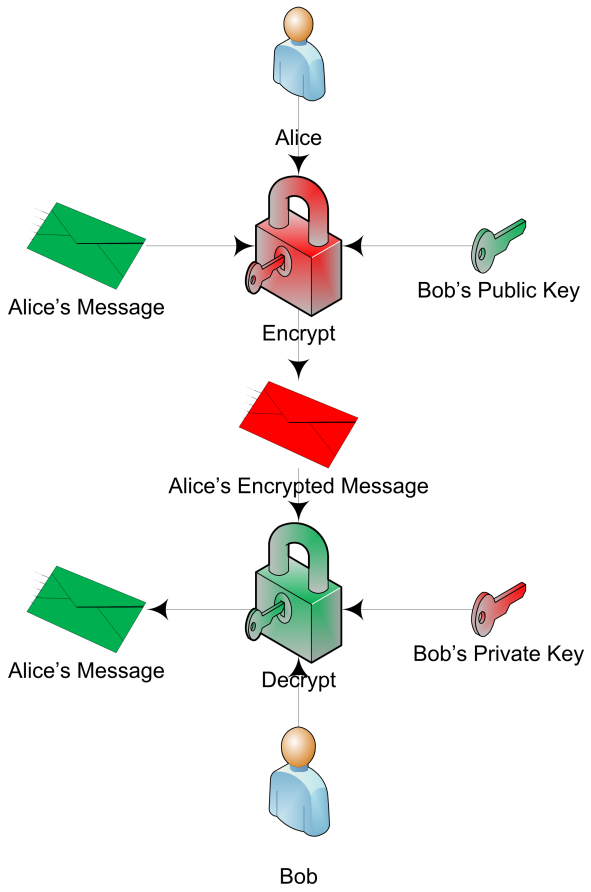

Figure 3.1: Public Key Cryptography Example

the decrypted certificate matches the original, thus asserting that the certificate is genuine. In the example, Alice must decide which CA's certificates she wishes to trust; once she does this, she can verify the validity of any certificate signed by that authority. [9]

3.1.3 Certificate Revocation

Continuing the example described in Figure 3.1, Alice gains additional confidence in the integrity of her communication when she has access to her trusted CA's Certificate Revocation List (CRL). From time to time, a certificate may be compromised by the loss or theft of a private key. When notified of this situation, the CA will revoke a certificate by placing it on the CRL. Since each valid certificate is unique, an entity wishing to validate a certificate may also check it against the publicly available CRL in order to determine if it has been revoked. A certificate appearing on a certificate revocation list will never be accepted, even if it passes all other checks. This allows the maintainers of the PKI to disallow certificates on a policy basis. [9]

3.2 Virtual Organizations

As mentioned above, one major component of OSG are the Virtual Organizations (VOs). A VO is a group of like-minded users who have joined together in order to share compute resources. The VO provides the public-key infrastructure, namely a set of CAs. Thus, by trusting the organization, the members implicitly trust each other. The origins of the grid are tied to traditionally compute-intensive disciplines such as high-energy physics who were quick to establish VOs. These VOs have existed since the beginning of the grid and as such, their users and administrators are well accustomed to grid software, processes, and policies. In recent years, as grid software has grown in complexity, so have the barriers to entry. Thus the need to a VO dedicated to the task of engaging new users has become apparent.[9]

3.2.1 Engagement

The Engagement VO (known colloquially as Engage) was created to acclimate new users to grid technologies and processes. It is tasked with bringing new users and resources to the grid, educating them, and allowing them to connect with like-minded individuals. The goal is for these users to then join another VO that suits their needs or start their own VO. Engage handles all

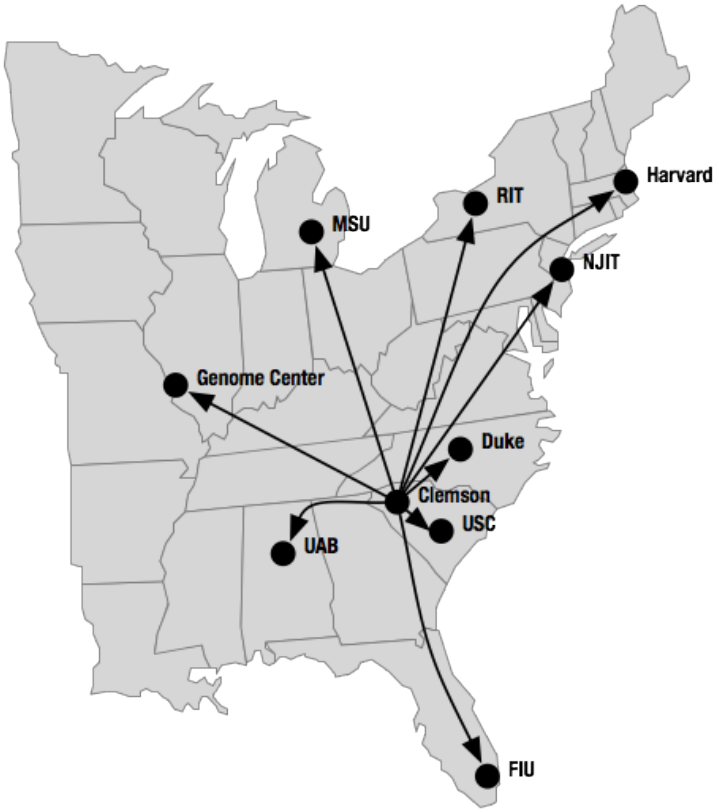

Figure 3.2: Clemson Ci-Team Activities

VO-level administration, including the PKI. This allows users to get up and running quickly and determine how to most effectively use grid technologies for their particular computational problems.

To further this goal, a cyberinfrastructure team or CI-Team was created at Clemson University. The team has assisted researchers at Duke University, the Harvard Medical School, the Rochester Institute of Technology, the New Jersey Institute of Technology, the University of South Carolina, the Washington University Genome Center, the University of Alabama at Birmingham, Florida International University, and Michigan State University (Figure 3.2) with the process of using the grid as well as the process of adding new resources to the grid. The Clemson CI-Team has also assisted with user education and resource deployment at Clemson University itself. [9]

3.3 Open Science Grid Sites

A Collection of resources is known as a site. Sites are the second main main component of the Open Science Grid and provide the actual computational and storage resources to VOs. Two main types of resources exist: Compute Elements (CEs) and Storage Elements (SEs). Since these resources are independent entities which users must interact with, they require their own certificates. These certificates, known as host certificates, must be granted by an established VO. This does not necessarily mean that a site trusts the users of the VO that granted its certificates, but this is generally the case. [9]

3.3.1 Compute Elements

A compute element consists of a Globus gatekeeper paired with a batch scheduling system such as PBS or Condor. Users can submit computational jobs to a compute element, and as long as the CE trusts that user, their jobs will run and the results will be returned to the user

Compute elements also commonly have site-local users who may or may not have grid identities and who interface directly with the back-end batch system. Since the grid software communicates with the batch system in the same manner as would any other user, this does not present an integration problem. [9]

3.3.2 Storage Elements

A storage element consists of a Storage Resource Manager (SRM) interface to a filesystem, and provides users with remote storage of data. Users can upload data to a storage element and then transfer this data directly to a compute element as input for a job. The user can orchestrate a workflow that involves transfers between CEs and SEs without ever having to use their local system as a staging area. This benefits users who may not have access to large storage arrays or high bandwidth connections in their local environment. The details of both the CE and SE software stacks are discussed in Chapter 4. [9]

3.4 Trust Model

The Open Science Grid is an organization bound together by mutual trust among federated virtual organizations. The three types of trust relationships are VO→User[1], Site→VO, and OSG→VO. These are discussed in detail below. [9]

3.4.1 VO→User Trust

A user wishing to use grid resources must join a virtual organization. The virtual organization establishes criteria for membership that must maintain a certain level of rigor, but OSG as an organization is not overly strict on this point. Essentially, each VO determines how to select its own membership. When a member is accepted into a VO, they receive a private key and certificate signed by the VO's CA(s). The user may then utilize any site which trusts their VO. [9]

3.4.2 Site→VO Trust

When new sites begin operations, the site administrators must decide from which VO's they will accept computational jobs. They will usually accept jobs from the VO which signed their host certificates, but this is not required. Factors that may influence this decision include requirements that a potential VO might have regarding compute power, software stack, number of concurrent users, and trust reciprocity. This last factor bears further examination supported by a key observation: a site is usually created by grid users or their affiliated institutions. While it is certainly possible for VOs not to reciprocate trust, this is not common. More commonly, a site will trust VO who trusts that site's sponsors. [9]

3.4.3 OSG→VO Trust

The third type of trust, whereby the central OSG organization decides to trust a VO, does not necessarily follow from the notion of a purely federated system. In fact, a VO can theoretically exist without the blessing of the central OSG organization; however, it is much more convenient if a central listing of VOs and their respective information is maintained. This way, the software stack can be distributed with VO management server addresses and CA certificates pre-installed, making a site administrator's job easier. [9]

[1] Note on notation: → is read as "trusts"

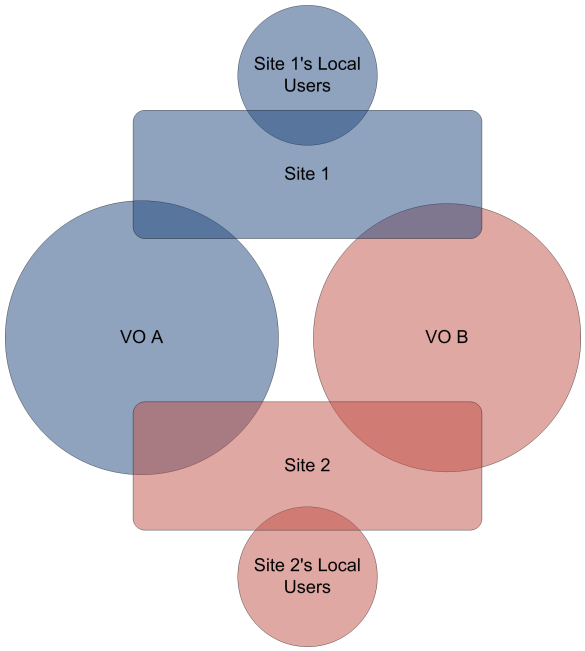

Figure 3.3: Sample Trust Scenario

3.4.4 Sample Trust Scenario

As an example (illustrated in Figure 3.3), suppose there exist two VOs, A and B. These VOs were established in order to allow their membership to pool computational resources. To that end, VO A has set up Site 1 located at Prestigious University, where many of VO A's members are employed. However, the administration at Prestigious University has decreed that these computational resources shall be available to all faculty and students. Thus, Site 1 has some local users who do not have grid identities as well as grid users, who may be physically co-located or remote. VO B is in a similar situation, and has thus set up Site 2. This state of affairs continues for a while, but the members of VO A and VO B begin facing tighter schedules and would like to have access to a higher throughput computing system. The two VOs decide to share resources, so Site 1 decides to begin trusting VO B and Site 2 begins trusting VO A. Now members of both VO A and B have two sites at their disposal and can choose which one they would like to use based on utilization patterns. The local users at each site may still only utilize their local resources, unless they too join a VO and obtain grid identities. [9]

Chapter 4

Open Science Grid Software Stack

Four distinct sets of software provide an implementation of a grid computing system for the Open Science Grid (OSG): user mapping software (Section 4.1), the Compute Element package (Section 4.2), the Storage Element package (Section 4.3) and monitoring and accounting software (Section 4.4). A short software use case is presented (Section 4.5). Some material in this chapter has been previously presented to the Calhoun Honors College of Clemson University as part of the author's undergraduate honors thesis.[9]

4.1 User Mapping Software

Once a user has been authenticated, their grid identity must then be mapped to a local user account with sufficient privileges to run their desired job. Two obvious ways to maintain this mapping would be to have an individual account for each grid user or to have one account to which all grid users are mapped. These are both unsatisfactory, but for differing reasons. Mapping each grid user to an individual account may seem promising, as it is the model is generally employed for local users, but it has its faults. Since VO membership constantly changes and user authentication policies and mechanisms differ among sites, developing an automated system generic enough to handle all cases would be a daunting task. Thus, it would fall to the site administrators to keep the mappings up to date, which is a waste of valuable systems administrator time. Likewise, the all-to-one mapping is advantageous due to its simplicity, but violates the basic trust model. A user process can generally inspect and manipulate all processes running as the same user. If all grid

Listing 4.1: Excerpt from a Grid-Mapfile

```
#—— members of vo: engage ——#
"/C=AU/O=APACGrid/OU=Monash University/CN=Blair Bethwaite" engage
"/C=AU/O=APACGrid/OU=Monash University/CN=Steve Androulakis" engage
"/C=MX/O=UNAMgrid/OU=DGSCA UNAM CU/CN=Eduardo Cesar Cabrera Flores" engage
"/C=UK/O=eScience/OU=Sheffield/L=CICS/CN=michael griffiths" engage
"/DC=es/DC=irisgrid/O=bsc-cns/CN=enric.tejedor" engage
"/DC=es/DC=irisgrid/O=bsc-cns/CN=jorge.ejarque" engage
"/DC=org/DC=doegrids/OU=People/CN=Abhishek Pratap 39489" engage
"/DC=org/DC=doegrids/OU=People/CN=Albert Everett 905390" engage
#—— members of vo: nanohub ——#
"/CN=Steven M Clark/OU=Purdue TeraGrid/O=Purdue University/ST=Indiana/C=US" nanohub
"/CN=nanoHUB Service00/OU=Purdue TeraGrid/O=Purdue University/ST=Indiana/C=US" nanohub
"/CN=nanoHUB Service01/OU=Purdue TeraGrid/O=Purdue University/ST=Indiana/C=US" nanohub
"/CN=nanoHUB Service02/OU=Purdue TeraGrid/O=Purdue University/ST=Indiana/C=US" nanohub
```

users were to be mapped to a single local account, any grid job could inspect or terminate any other job, regardless of VO. A single VO has made a decision to trust its own members, but may not necessarily trust any other VOs.

The solution to this problem lies within the basic organization of the Open Science Grid. Since each VO member trusts all other members, and does not necessarily trust the members of any other VO, there must be at least one local account per VO. On the other hand, since site administrators make trust decisions at the VO level, they should not have to worry about the individual membership of a VO. Thus local VO accounts should not make distinctions based on individual users. Therefore, the model of having one local user account per VO is shown to be sufficient. [9]

4.1.1 Grid-Mapfile

The most basic way to actually map grid users to local accounts is by using a grid-mapfile. This file is a simple key-value pairing of grid identities to local user accounts. This file is updated automatically by a component of the OSG software stack (edg-mkgridmap). For each VO that the site trusts, a list of members will be downloaded from each VO's central server at a given interval. These will then be paired with the configured local user account and written to the grid-mapfile.[9]

Listing 4.1 contains an excerpt from a gird-mapfile. Note that even though grid identities should be issued to a particular person due to accountability requirements, many VOs will also issue certificates to middleware.

4.1.2 Grid User Mapping System

The grid-mapfile approach works well and has the advantage of being simple. However, it is not the most efficient solution when considering the case of multiple resources within the same administrative domain. Instead of having each resource maintain its own mappings, it would be more efficient to have a central mapping server provide user mappings to all resources within an administrative domain. This is exactly the functionality that the Grid User Mapping System (GUMS) provides. When a site utilizes GUMS, only the GUMS server needs to contact each VO's server and download the current membership roster. Individual resources can then contact the GUMS server whenever a job submission occurs. The GUMS server can then respond with a mapping, eliminating the need for a grid-mapfile to be present at each resource. [9]

4.2 Compute Element Software

As mentioned in Chapter 3, the two main components of a Compute Element (CE) are the Globus gatekeeper and the back-end batch system. The gatekeeper is the grid interface to the batch system and thus only it needs to be publicly accessible and possess a grid certificate. The batch system can be located solely on a private network, indeed, this is how many dedicated computational resources are configured. [9]

4.2.1 The Globus Toolkit

In the OSG environment, the Globus toolkit serves two primary functions. It handles grid user authentication and serves as the interface to the batch system. In its user authentication role, Globus handles the public-key cryptography and certificate validation discussed above. The OSG software distribution includes a mechanism to keep the Globus's local copies of the CA certificates and CRLs up do date.

Once a grid identity has been successfully mapped, the Globus toolkit can then service the request. Two main types of services are provided by Globus: GRAM and GridFTP. GRAM is a

mechanism for running commands non-interactively at a remote site, while GridFTP is a grid-aware file transfer program. Users typically use GridFTP to "stage-in" data sets, then use GRAM to fork and execute the job directly or invoke the batch system. Finally, the users use GridFTP to retrieve the job's output. Globus interfaces to batch systems are referred to as job managers and will be discussed in detail below.

Also of note is WS-GRAM. Globus was originally designed around a custom session, presentation, and application protocol. WS-GRAM is a new version of GRAM which seeks to replicate and extend the functionality of this protocol, but through a standard, web-services communication model. [9]

4.2.2 Job Managers

The job managers provided with Globus provide an interface to several popular batch execution systems including Condor, the Portable Batch System (PBS), and the Sun Grid Engine (SGE). The simplest of these job managers is known as the Fork job manager. Fork is essentially a null job manager, because it simply forks and executes the job on the current machine, i.e. the compute element itself. Fork is ideal for short-lived jobs and simple diagnostics, but suffers from a critical flaw. Since Fork is stateless, a malicious user could perform a "fork bomb" attack on a compute element, overwhelming it with processes and eventually crashing the machine. Due to this vulnerability, site administrators are encouraged to use the Managed-Fork job manager, which uses Condor to limit the number of running processes

The Condor job manager interfaces with the popular Condor High Throughput Computing system. The job manager handles the creation of the Condor submission script and also retrieves the output from Condor. It is important to note that the job manager does not posses a direct interface to Condor, and instead interacts with the system in the same way that an end user would. The PBS and SGE job managers perform similarly, with regard to their respective batch systems. [9]

4.3 Storage Element Software

The second main type of resource that a site can provide is the Storage Element (SE). The SE provides a Storage Resource Management (SRM) interface to a storage array. SRM is grid-

aware in the sense that it can map grid identities to a VO-specific storage pool. These pools are securely partitioned among VOs, thus maintaining the trust model. The two main implementations of the SRM interface are dCache and BeStMan. DCache provides an implementation of the SRM interface tightly coupled to a large-scale distributed filesystem. BeStMan can provide an SRM interface to any filesystem. The only requirements are that the filesystem be mountable on the SE node and have typical UNIX-style file ownership and permissions. The main advantage that dCache has over BeStMan is scalability. With BeStMan, while the underlying filesystem may indeed be distributed in nature, the SE head node becomes a bottleneck for network traffic. Under dCache, each storage node is grid-aware, and can perform network transfers independently, thus avoiding the bottlenecking problem. DCache has a downside in that it requires a significant number of dedicated metadata nodes and thus does not scale down to small deployments very well. [9]

4.4 Monitoring and Accounting Software

Finally, OSG maintains a suite of monitoring services and information providers. These include service advertisement, usage reporting, and diagnostic tools. These capabilities are historical weak points of OSG, due to the federated nature of its services. However, development is in progress, with the goal of improving upon these weak points. Two of the most useful monitoring and accounting systems are the Resource and Service Validation (RSV) system, and the Gratia accounting system.

The primary site monitoring system for OSG is RSV. RSV operates on each site by running a set of periodic jobs against the site. These jobs interact with the site in the same way that a user would for the purpose of providing a complete end-to-end test. Each test, or *probe*, is reports its results to the site administrator and to the central OSG organization. Commonly installed probes include monitoring of the status of the default job manager, the list of available job managers, the site's CA certificate package version, the site's CRL expiration dates, the permissions on the local storage pool, the OSG software version, a basic ping test, the VDT version, which VOs are supported, the status of the Globus GRAM service, the status of the GridFTP service, the status of the site's batch scheduler, the status of the GUMS server, and the expiration dates of the site's host certificates. RSV maintains a detailed history of these probes' results to both the site administrator and OSG central organization. These results can be particularly useful when troubleshooting a new site.

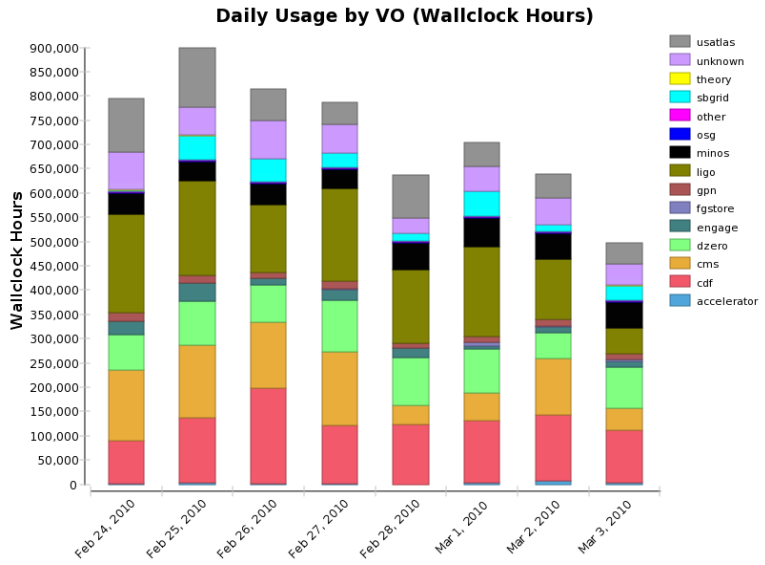

Figure 4.1: Gratia – Daily Usage by VO

The primary accounting system in the OSG is Gratia. Gratia can give very detailed reports on utilization on a per-site basis, or for the grid as a whole. Gratia works by inserting probes into the OSG software stack that report exact usage numbers back to a central Gratia web service. Gratia does not provide a real-time view of the grid because its goals are accuracy and completeness of records.

Gratia provides information on wall time, processor time, and job counts. This information can be provided over any date range and interval desires. This data is visualized via engaging charts and graphs, one of which is presented in Figure 4.1. [9]

4.5 Software Use Case

To illustrate how the OSG software stack interacts with users and functions as a distributed system, an example use case will now be presented, illustrated with Figure 4.2.

Suppose a user wishes to run a scientific job on the Open Science Grid. He first looks at the central RSV repository and compiles a list of sites that fit his requirements. He decides to use Compute Element A and Compute Element B. Since his job requires a large amount of intermediate storage, and he does not have a high-bandwidth connection in his local environment, he decides that he would like to use Storage Element C as an intermediate storage location. CE A uses GRAM,

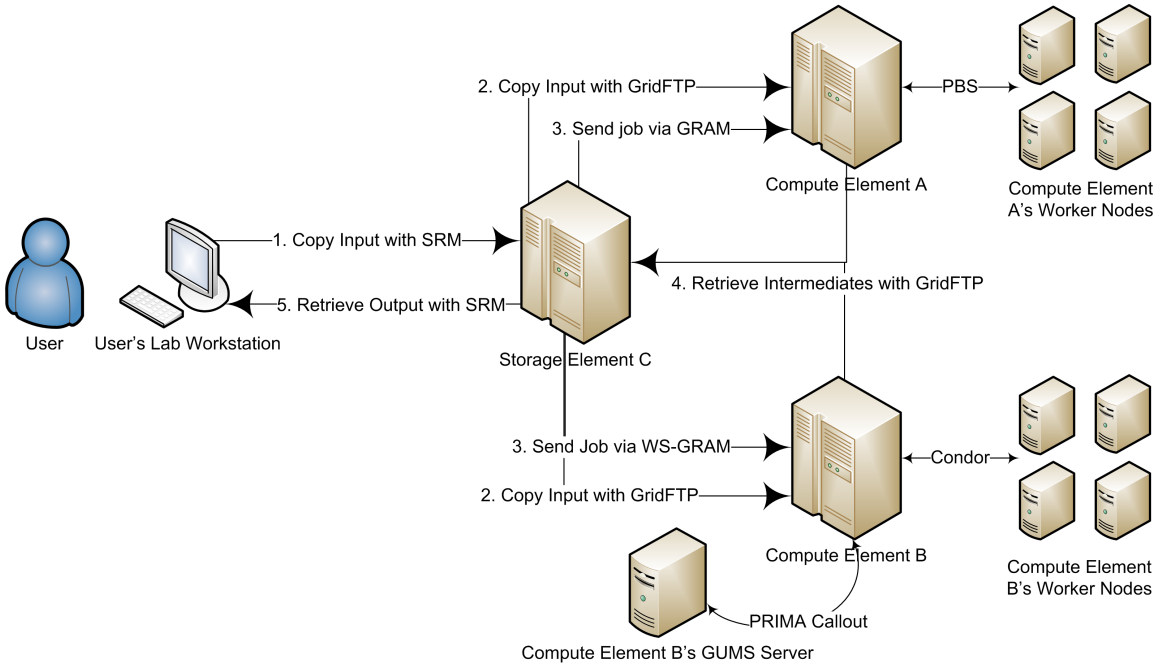

Figure 4.2: Grid Software Use Case

a grid-mapfile, and the PBS batch system, while CE B uses WS-GRAM, a GUMS server, and the Condor batch system.

To begin, the user copies his input data to SE C. He then uses GridFTP to copy the relevant portions of the data to CEs A and B. Once his data has been *staged-in* to CEs A and B, he sends his job via GRAM to CE A and via WS-GRAM to CE B.

The user's job request to CE A contains a copy of his certificate, which is authenticated by Globus. His grid identity is then be mapped to the local user account for his VO by the grid-mapfile. His job is then submitted to PBS where it begins executing. Gratia monitors the job and reports its execution time to the central Gratia repository.

Meanwhile, the user's job request to CE B has also been authenticated by Globus, and his grid credentials are being processed by the GUMS server. GUMS returns a user mapping to the CE which then submits his job to the Condor batch scheduler, where it begins executing. Gratia also records this jobs elapsed time.

After the user's jobs run, he then copies the intermediate results back to SE C via GridFTP. He can then prepare the next iteration of his job or retrieve the results from the SE. [9]

Chapter 5

Virtual Organization Cluster Model

The Virtual Organization Cluster (VOC) Model specifies the high-level properties of systems that support the assignment of computational jobs to virtual clusters. This chapter has been published by Murphy, Fenn, and Goasguen in the 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009) [8]. Some material in this chapter has also been presented to the Calhoun Honors College of Clemson University as part of the author's undergraduate honors thesis.[9]

It is important to note that each VOC is solely dedicated to an individual VO. However, multiple virtual clusters can be present on a single physical cluster at the same time. A fundamental division of responsibility between the administration of the physical computing resources and the virtual machine(s) implementing each VOC is fundamental to the VOC Model. For clarity, the responsibilities of the hardware owners are said to belong to the Physical Administrative Domain (PAD). Responsibilities delegated to the VOC owners are part of the Virtual Administrative Domain (VAD) of the associated VOC. Each physical cluster has exactly one PAD and zero or more associated VADs. VADs are not necessarily unique to a particular PAD, as a virtual cluster may span multiple physical clusters. [8, 9]

Figure 5.1 illustrates an example system designed using the VOC Model. In this example, the PAD contains all the physical fabric needed to host VOCs and connect them to the grid. Each physical compute host in the PAD is equipped with a virtual machine monitor for running VOC nodes. Shared services, including storage space, a grid gatekeeper, and networking services are also provided in the PAD. Two VOCs are illustrated in Figure 5.1, each having its own independent

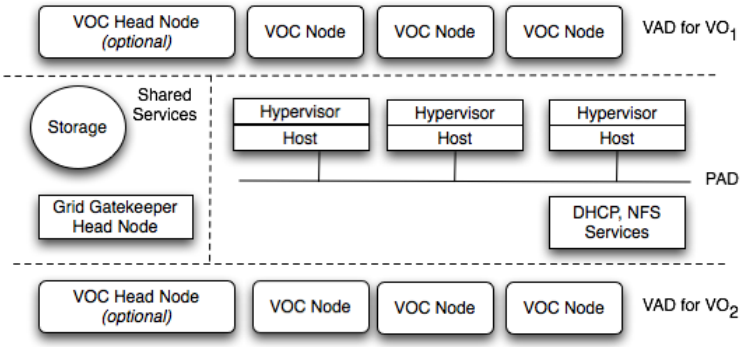

Figure 5.1: PAD and VAD

VAD. Each VOC optionally includes a virtual head node that, if present, receives incoming grid jobs from the shared gatekeeper in the PAD. Alternatively, each VOC node can receive jobs directly from the shared gatekeeper, by means of a compatible scheduler interface. The PAD administrator must make certain allowances for these VOC head nodes, in particular, he or she must provide inbound and outbound network connectivity to nodes on an as-specified basis. [8, 9]

In practice, Virtual Organization Clusters can be supplied by the same entity that owns the physical computational resource, by the Virtual Organizations (VOs) themselves, or by a contracted third party. Similarly, physical fabric on which to run the VOCs could be provided either by the VO or by a third party. One possible model for third-party physical system providers is that of a Virtualization Service Provider (VSP). A VSP offers clusters of hardware configured to support VMs, along with networking and Internet connectivity for those VMs. VOs can contract with VSPs to provide the necessary infrastructure for hosting VOCs, avoiding the requirement for each VO to invest in infrastructure such as hardware, power, and cooling. This abstraction of compute resources is a key objective of grid computing. In turn, VSPs offer VM hosting services to multiple VOs, perhaps employing time-based or share-based scheduling to multiplex VOCs on the same hardware. [8, 9]

5.1 Physical Administrative Domain

The Physical Administrative Domain (PAD) contains the physical infrastructure (see Figure 5.1), which is comprised of the host computers themselves, the physical network interconnecting those hosts, local and distributed storage for virtual machine images, power distribution systems,

cooling, and all other infrastructure required implement a physical cluster. Also within this domain are the host operating systems, virtual machine monitors, and central management systems for physical servers. Fundamentally, the hardware cluster provides the Virtual Machine Monitors (VMMs) needed to host the VOC system images as guests. [8, 9]

An efficient physical cluster implementation requires some mechanism for creating multiple compute nodes from a single disk image submitted by the VO. One solution is to employ a VMM with the ability to spawn multiple virtual machine instances from a single image file in a read-only mode that does not persist any changes made at run-time to the image file. Another solution is to use a distributed file copy mechanism in order to replicate local copies of each VM image to each execution host. Without this type of mechanism, the VO would be required to submit one VM image for each compute node, which would result in both higher levels of Wide Area Network traffic and greater administrative difficulty. Thus, such a mechanism is considered to be essential to the VOC Model. [8, 9]

Various architectures may be employed to design physical systems that provide the necessary virtualization resources to guests. One simple architecture would utilize commodity rack-mount server hardware to provide raw computational power, with standard networking components providing system interconnects. A basic Linux system with a VMM can provide virtualization services, while standard networking services, such as the Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) servers, would be provided by dedicated physical hosts. Guest virtual machines in such an architecture would thus be indistinguishable from physical hosts: the virtual machines would be provided with networking services as if they were physical hosts. Arbitrary guest operating systems can be supported as long as the Instruction Set Architecture (ISA) of the guest is compatible with the ISA of the host. Alternatively, paravirtualized guests could be supported with a paravirtualization system, either employing direct use of physical network resources or contained within a separate virtual networking environment. With a paravirtualization system, the guests would have to be configured to make use of the paravirtualized hardware. Thus, certain operating systems cannot be supported as paravirtualized guests. [8, 9]

Optionally, the physical resource provider may supply common interfaces to shared resources with which the hosted virtual machines might interact. For example, a physical resource might provide a common gatekeeper for all the hosted VOCs, which could provide a connection to a grid

such as the Open Science Grid. Other examples of shared resources might include shared storage space, a common job scheduling system, or a shared virtual gateway for server connections. [8, 9]

5.2 Virtual Administrative Domain

Each Virtual Administrative Domain (VAD) consists of a set of virtual machine images for a single Virtual Organization (VO). A VM image set contains one or more virtual machine images, depending upon the target physical system(s) on which the VOC system will execute. In the general case, two virtual machine images are required: one image of a head node for the VOC, and another image that is used to spawn all the compute nodes of the VOC. When physical resources provide a shared head node, only a compute node image with a compatible job scheduler interface is required. [8, 9]

Perhaps the greatest challenge for the VAD administrator is the requirement a single compute node VM image may be used to spawn multiple VM instances. In other words, the image must be configured in such a way that it can be *contextualized* for each VM when instantiating multiple VMs [41]. No assumptions about the size of the VOC, the type of networking, the hostname of the system, or any system-specific configuration settings should be stored in the image. Instead, standard methods for obtaining network and host information (e.g. DHCP) should be used, and any per-VM-instance configuration should be made dynamically at boot time. [8, 9]

VMs configured for use in VOCs may be accessed by the broader grid in one of two ways: If the physical fabric at a site is configured to support both virtual head nodes and virtual compute nodes, then the virtual head node for the VOC may function as a gatekeeper between the VOC and the grid, using a shared physical grid gatekeeper interface as a proxy. In the second case, the the single virtual compute node image needs to be configured with a scheduler interface compatible with the physical site. The physical fabric then provides the gatekeeper between the grid and the VOC (Figure 5.1), and jobs are matched to the individual VOC. [8, 9]

5.3 Provisioning and Execution of Virtual Machines

Virtual Organization Clusters are configured and started on the physical compute fabric by middleware installed in the Physical Administrative Domain. Such middleware can either receive a

pre-configured virtual machine image (or pair of images) or provision a Virtual Organization Cluster on the fly using an approach such as In-VIGO [4], VMPlants [5], or installation of nodes via virtual disk caches [19]. Middleware for creating VOCs can exist directly on the physical system, or it can be provided by another (perhaps third-party) system. To satisfy VAD administrators who desire complete control over their systems, VM images can also be created manually and uploaded to the physical fabric with a grid data transfer mechanism such as the one depicted in the use case presented in Figure 1.1. [8, 9]

Once the VM image is provided by the VO to the physical fabric provider, instances of the image can be started to form virtual compute nodes in the VOC. Since only one VM image is used to spawn many virtual compute nodes, the image must be read-only. Run-time changes made to the image are stored in RAM or in temporary files on each physical compute node and are thus lost whenever the virtual compute node is stopped. Since changes to the image are non-persistent, VM instances started in this way can be safely terminated without regard to the machine state, since data corruption is not an issue. As an example, VM instances started with the KVM virtual machine monitor are abstracted on the host system as standard Linux processes. These processes can be safely stopped (e.g. using the SIGKILL signal) instantly, eliminating the time required for proper operating system shutdown in the guest. Since there is no requirement to perform an orderly shutdown, no special termination procedure needs to be added to a cluster process scheduler to remove a VM from execution on a physical processor. [8, 9]

When booting a VM instance from a shared image, certain configuration information must be obtained dynamically for each instance in order to contextualize the instance. In particular, each virtual compute node requires network connectivity to enable communications. For most purposes, a virtual compute node can be treated as a physical node that has been shipped to the physical fabric site by the VO: the virtual compute node can simply use existing dynamic protocols to obtain a network address and network connectivity. However, an issue does arise with the shared read-only image model, in that the Media Access Control (MAC) address of each VM instance needs to be unique on the local network. A solution to this problem is to treat the MAC address as a dynamic resource that is leased to each VM instance (Figure 5.2). Thus, MAC addresses become part of the PAD and are a resource managed by the supporting middleware. Other resources that need direct mapping to physical devices or low-level protocols will need to be leased to the VOCs in a similar fashion. Such resources should be dynamically allocated by the middleware and given directly to

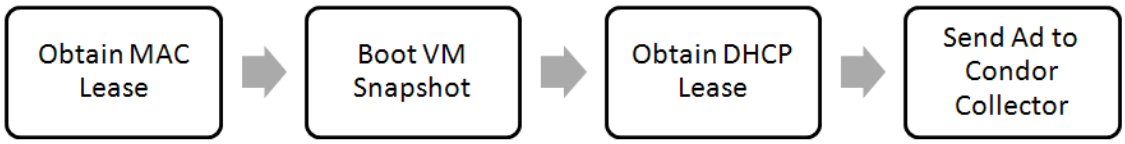

Figure 5.2: VOC node Boot Process

the virtual machine monitor, which abstracts the resource as virtual hardware. In effect, the guest operating system should be unaware that a low-level resource like a MAC address is leased and not owned. [8, 9]

Once mechanisms are in place to lease physical resources and start VMs, entire virtual clusters can be started and stopped by the physical system (Figure 5.3). VOCs can thus be scheduled on the hardware following a cluster model: each VOC would simply be a job to be executed by the physical cluster system. Once a VOC is running, jobs arriving for that VOC can be dispatched to the VOC. The size of each VOC could be dynamically expanded or reduced according to job requirements and physical scheduling policy. Multiple VOCs could then share the same hardware using mechanisms similar to those employed on traditional clusters. [8, 9]

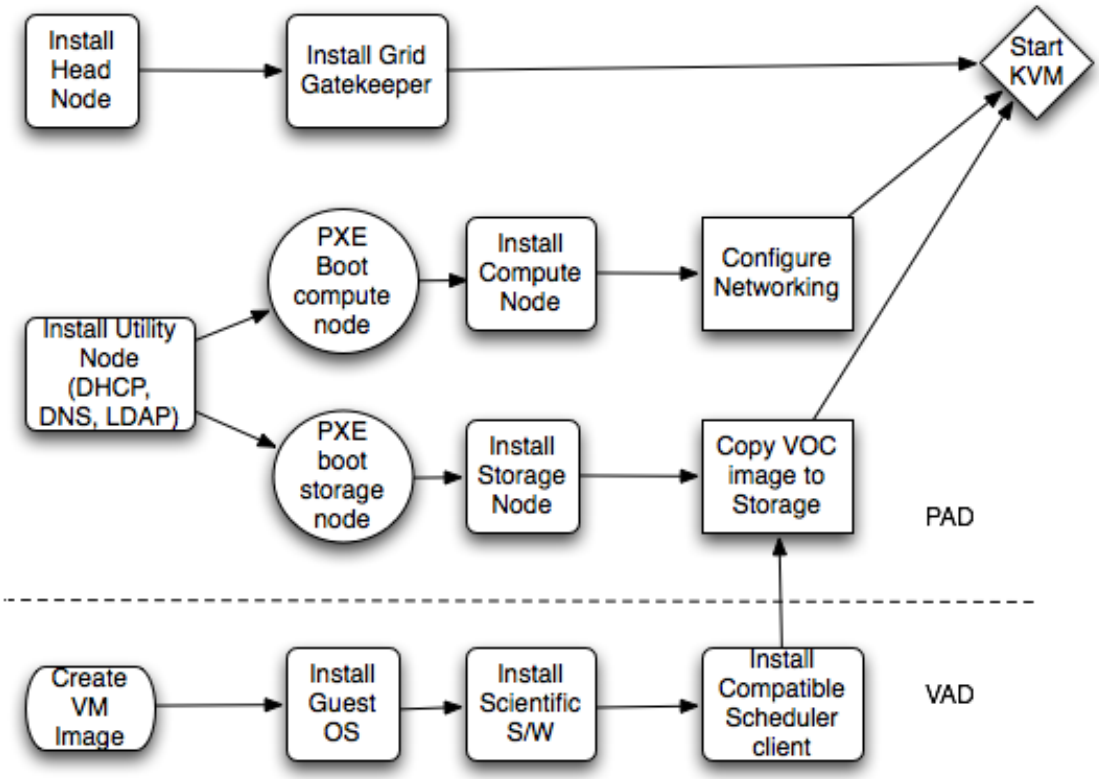

Figure 5.3: Ideal Cluster Provisioning Process

Chapter 6

Implementation of the Virtual Organization Cluster Model

The core focus of the body of research presented in this thesis is the implementation of the Virtual Organization Cluster Model. Important considerations of this implementation include the details of how the virtual compute nodes are created and instantiated (Section 6.1) as well as the how the physical support services are implemented (Section 6.2). Dynamic provisioning of the VOC is implemented via the watchdog mechanism discussed in Section 6.3. VOCs which span NAT boundaries need some sort of overlay network. The prototype VOC was implemented with the IPOP overlay networking solution, discussed in Section 6.4. Some material in this chapter has been previously presented to the Calhoun Honors College of Clemson University as part of the author's undergraduate honors thesis as well as in the 47th ACM Southeast Conference. [9, 42]

6.1 Virtual Cluster Construction

An essential component of any virtual cluster is the choice of Virtual Machine Monitor (VMM). This choice will have far-reaching implications in terms of restricting future configuration options, *i.e.* host operating system and performance goals. Two such VMMs are compared, the Kernel-based Virtual Machine and Xen, in Sections 6.1.1 and 6.1.2 respectively. The configuration of the virtual compute nodes is discussed in Section 6.1.3, and their deployment to the Open Science

Grid is described in Section 6.1.4. A discussion of the tools and techniques used to contextualize VM images is presented in Section 6.1.5. [9, 42]

6.1.1 Kernel-based Virtual Machine (KVM)

KVM is an extension of the well-known QEMU emulator with support for the x86 VT extensions [43]. These extensions allow virtual machines to make system calls without unnecessarily invoking the host kernel, thus potentially saving two context switches [44]. KVM requires a recent Linux kernel with the KVM modules enabled. This virtualization system differs from other virtualization technologies that require heavily modified kernels, whose development is not often concurrent with the mainline kernel. Prototype implementation began with KVM-57, but issues with the emulated network interface card (NIC) compelled an upgrade to KVM-77.

KVM inherits all QEMU tools, thus supporting the QEMU Copy-on-write (QCOW) disk image format. QCOW supports a *snapshot* mode for disk I/O in which all disk writes are directed to a temporary file and are *not* persisted to the original image. This mode allows multiple VMs to be run from a single master disk image mounted from a network location. Such an arrangement mitigates the storage requirements associated with running a cluster of VMs [45]. *Snapshot* mode also makes destroying a running virtual cluster as simple as sending SIGKILL to the virtual machine monitor on each physical node.

Also inherited from QEMU is the ability to use the TUN/TAP Ethernet bridge available in the Linux kernel. The bridge essentially emulates a switch, allowing each VM to have individual networking resources, separate from both the host and other VMs. Each TAP device acts as a virtual Ethernet endpoint, each connected to a software bridge, along with the hardware Ethernet endpoint as shown in Figure 6.1. [9, 42]

6.1.2 Xen

Xen is a hypervisor and is similar to KVM in the sense that it allows multiple guest operating systems, domains in Xen parlance, to run concurrently on the same hardware [46]. Xen, however differs from KVM in that it does not rely on any type of CPU instructions for virtualization support, instead it uses a technique known as *paravirtualization* [47]. In this technique, the guest operating system (OS) is modified in such a way that all supervisor instructions (those that would, in KVM,

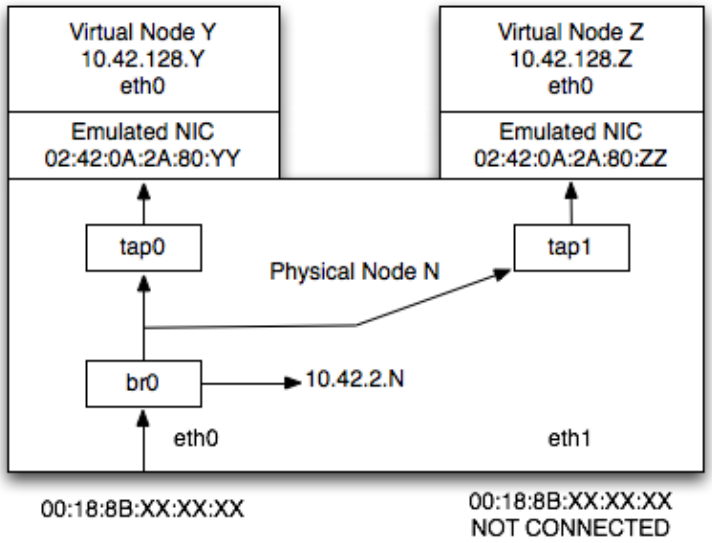

Figure 6.1: VOC node Bridging

be handled by the VT extensions) are replaced by *hypercalls* into the Xen hypervisor. This allows for much greater performance than pure emulation (QEMU) and competitive performance with VT solutions such as KVM. [9, 42]

6.1.3 Virtual Compute Nodes

Each VOC is composed of VMs known as VOC nodes. Every VO that wishes to use a VSP's computational power must submit a VOC node image (or set of images) to the VSP, along with some configuration parameters. The VO must carefully construct the VOC node image, since each image could be used to start multiple VMs. To this end, a VOC node image must not make any assumptions about the network hardware, hostname, or other system-specific settings. Dynamic configuration must be used instead.

When booting a VOC node, the virtual machine monitor must first obtain a MAC address, then boot the VM in snapshot mode. One the VOC node OS begins to boot, it will receive a Dynamic Host Configuration Protocol (DHCP) lease for its IP address and then join a scheduling pool, such as Condor. [9, 42]

6.1.4 Grid Integration

The prototype VOC is built from two CentOS 5 VM images: the VOC node image and an OSG gatekeeper image that can be shared among multiple VOCs. CentOS is a good choice of operating system because of its extensive support for cluster and scientific computing software. The virtual head node is configured as an OSG gatekeeper through installation of the Virtual Data Toolkit (VDT), Condor central manager and submit daemons, the Ganglia monitoring daemon, and the Ganglia metadata daemon. The virtual compute node is configured with the Condor starter daemon, MPICH2, ATLAS, and the Ganglia monitoring daemon. The prototype VOC is compatible with the OSG, and has been successfully deployed to OSG Integration Testbed as the Clemson-Birdnest site. All VOC nodes are Condor execute nodes and form a pool managed by the virtual head node running Globus GRAM. [9, 42]

6.1.5 VM Contextualization

The topic of VM contextualization merits further discussion. It is a safe assumption that any given VM image will not successfully integrate into a VOC as implemented at any given site. The image must be contextualized in two phases: image-level and instance-level. Image-level contextualization occurs once per VM disk image per site. Instance-level contextualization occurs once per VM instance. [48]

Image-level contextualization

Important considerations for image-level contextualization are image format, image layout, shared filesystem support, and batch scheduler integration. Image format refers to the representation of the disk's data within the image file. Image layout refers to how the various partitions are placed on the disk and to what other disk structures are present.

The simplest image format is that of the raw disk image. A raw image is simply a file containing the exact byte string that would appear on a physical device. This format is highly compatible but is not space efficient because the image file's size must be equal to the capacity of the virtual device being represented. Note that raw images compress very well with gzip compression, so they are fairly easy to distribute. In order to mitigate the in-use size issue, there has been a proliferation of virtual image formats such as VMDK, VDI, VHD, and QCOW2. These formats

vary in implementation and hypervisor support, but they all allow the compact representation of a disk image. When utilizing one of these formats, the size of the image is determined by the size of actual data present on the device, instead of by the capacity of the device. In order to contextualize the VM image format, the image must simply be converted to a format that is compatible with the hypervisor used at a given site. The `qemu-img` tool provides conversion functionality that can convert images between many of the popular formats. Hypervisor vendors also generally provide a tool that can convert between their format and the raw format.

The image layout issue can become much more involved. The two main image layouts are the partition image layout and the disk image layout. A partition image contains a representation of a single disk partition. Essentially, this layout could be referred to as a filesystem image, since a partition does not contain any metadata with regard to itself. This layout requires a hypervisor that is able to present individual partitions to a guest OS. Currently, only the Xen hypervisor is capable of this. The disk image layout contains a representation of an entire disk, including the master boot record, boot sector, and partition table. All hypervisors, including KVM, are capable of utilizing this type of image. Since Xen requires the guest kernel and initial ramdisk to be located outside of the VM image, Xen may only boot from disk images when it is used in conjunction with the `pygrub` utility. This utility mounts the disk image and extracts the kernel and initial ramdisk from the image, and as such, can only be utilized with a disk in the raw disk format. There is no set procedure for converting between partition images and disk images. Images will generally need to be converted (at least temporarily) to the raw format in order to allow standard disk tools to be utilized. There are, however, several useful tools and one guiding principle. The principle is: a disk image is the same as a physical disk, and a partition image is the same as a physical partition. Converting between image formats is a matter of getting the correct disk structures into the correct places. Useful tools include:

- `fdisk`, allows the calculation of partition extents and the creation/modification of partition tables,
- `dd`, allows block level copying of defined sections of an image,
- `mount`, when used with the `-o loop` option allows a partition image to be mounted,
- `kpartx`, allows the exposure of the partitions of a disk image as individual devices,

- `chroot`, allows the running of the native tools present in the image if necessary.

These tools, along with the bootloader installer, should be sufficient to assemble a disk image from a set of partition image or decompose a disk image into a set of partition images. An example of converting a partition image to a disk image is maintained by the author [49].

As the OSG compute node specification requires that various filesystems be shared among the compute element and its associated worker nodes, the image must also be contextualized so that it properly mounts those filesystems. In particular, any software libraries needed to mount the site's shared filesystem must be installed and the `$OSG_APP`, `$OSG_DATA`, and `$OSG_GRID` shares must the mounted in the locations defined by the CE configuration.

There must also be a way to get computational jobs into the VM. Either the site's batch scheduler or a VO-level scheduling system must be installed into the VM image. If the site's batch scheduler is installed, it is prudent to configure the scheduling system in such a way that the VM's scheduling pool may be partitioned off from the site's general scheduling pool in order to satisfy the constraints of the VOC Model. If a VO-level scheduler is installed, some provision must be made for crossing NAT boundaries. For further discussion of this point, see Section 6.4. [48]

Instance-level contextualization

Whereas image-level contextualization can be performed manually by a systems administrator, instance-level contextualization occurs once per VM instantiation and as such must be automated. As described in Section 5.3, certain resources must be leased from the physical site. These resources include network addresses, disk space, and scheduler slots.

Network addresses, including both MAC and IP addresses, should be assigned (leased) to the VMs in such a way as to avoid conflicts. Leasing of MAC addresses must be performed by the hypervisor. Leasing of IP addresses may be performed by the hypervisor if it is capable of passing this information to the guest (e.g. Xen) or may be through the standard DHCP protocol. One such method of assignment is to implement a central leasing server. Before VM instantiation, the hypervisor node would contact a central service and made a lease request for a MAC or IP address. The service would then maintain a lease database in order to avoid duplication. Since MAC and IP addresses will be unique to a hypervisor node, that node may also use a function to map its address to that of the VM. As long as this function will not cause an overlap in addresses, this method satisfies the uniqueness constraint without the requirement of a centralized service.

If the VOC nodes are not spawned from a single image, some allocation of disk space must be made to the hypervisor. This could use hypervisor's local disk, but care must be taken to avoid exceeding the disk's capacity, especially when dynamically resizing disk image formats are used. Another solution would be to map LUNs of a storage area network to the hypervisor node.

If the scheduling system requires the use of fixed slots for compute nodes, then these must also be assigned [35]. Techniques described for leasing network addresses can be easily extended to provide for such a scheduler. [48]

6.2 Physical Support Model

The physical cluster consists of seventeen Dell PowerEdge 860 1U rack-mount systems and one Dell PowerEdge 2970 2U rack-mount server. Each PowerEdge 860 machine is configured with a 2.66 GHz dual-core Intel Xeon CPU, 4 GiB of Double Data Rate 2 (DDR2) memory, and an 80 GB hard disk drive. The 2U PowerEdge 2970 server was configured with three 250 GB hard disk drives in a Redundant Array of Independent Disks, level 5 (RAID-5) configuration that hosts installation images, user home directories, network services, and a shared VM image store exported via a Network File System server. One of the 1U nodes is used both as a physical head node and as a host for the shared virtual head node. The other sixteen 1U nodes host two VOC nodes each. For an overview of the VOC layout, see Figure 6.2. [9, 42]

6.2.1 Host Operating System Configuration

CentOS 5 is a good choice for the host Operating System (OS) for many of the same reasons as it is for the VOC node OS. Sharing of software packages is an additional benefit of having a homogeneous OS environment. While CentOS does ship with KVM support, building KVM packages from the most recent sources allows the leveraging of the rapid pace of KVM development.

All compute nodes are installed via a Red Hat-style kickstart script, with some custom additions. Since kickstart installations can only utilize packages from a single repository, a custom addition to the kickstart system that allows packages from multiple repositories to be installed is utilized. [9, 42]

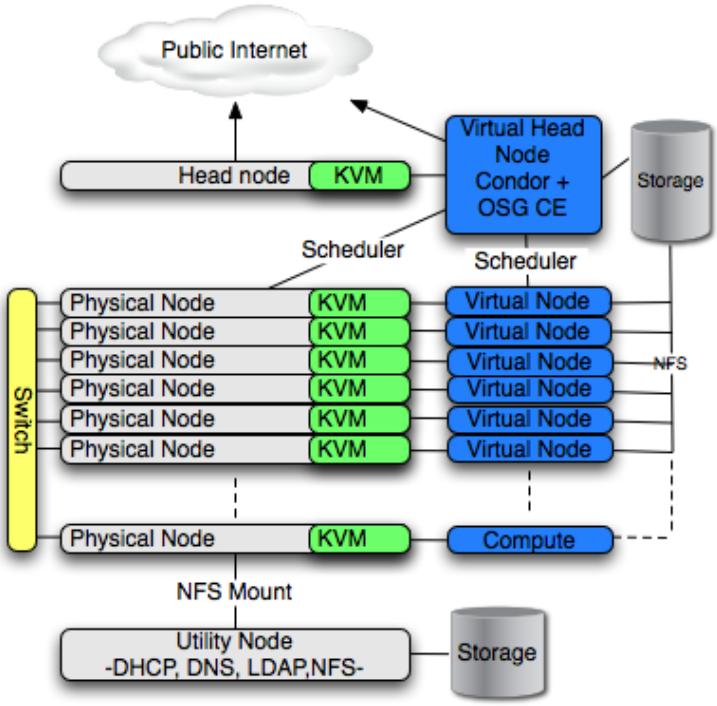

Figure 6.2: VOC Organization

6.2.2 Physical Support Services

A Lightweight Directory Access Protocol (LDAP) server is used as a central repository of configuration information for both the physical nodes and VOC nodes, including: hostnames, Internet Protocol (IP) addresses, and Media Access Control (MAC) addresses. VOC node MAC addresses are generated as locally-administered addresses to avoid any potential conflicts on the local subnet. To aid in administration of the physical nodes, an LDAP-aware, batch, remote administration tool is used[50].

The PowerEdge 2970 utility system provides a Domain Name System (DNS) server (dnsmasq) and a Dynamic Host Configuration Protocol (DHCP) server (ISC DHCPD). To maintain a single configuration source, utilities that generate DNS and DHCP configuration files dynamically from LDAP are employed. All VOC node images are hosted on a Network File System (NFS) export from the utility system. [9, 42]

6.3 Dynamic Provisioning

Dynamic provisioning refers to the autonomic starting and stopping of VMs in response to load. The mechanism to conduct this provisioning is known as a *watchdog*. The watchdog uses the Condor queue as its data source. It observes on a per-VO basis, the number of jobs waiting to be executed. When the number of jobs for a particular VO exceeds the size of that VO's VOC, the watchdog attempts to start a new VM. The maximum size of any VOC is determined by the systems administrator. Conversely, when the watchdog observes the number of jobs for a particular VO fall below the size of that VO's VOC, the watchdog will terminate a VM. The minimum number of VMs is $\max(0, x)$ where x is a value determined by the systems administrator. It is useful to set a minimum VOC size above zero because short jobs may otherwise be inordinately delayed. [51]

6.4 Overlay Networking

Most current batch schedulers cannot schedule jobs across a pool of worker nodes that exist behind separate Network Address Translation (NAT) gateways. In order to overcome this limitation, an overlay network must be provided to allow each worker node (VMs in the case of VOCs) to appear to be in a single subnet.

One such solution is IPOP, was discussed in Chapter 2. IPOP employs specific algorithms to allow direct connections to be made across NAT boundaries. IPOP differs from normal Simple Traversal of User Datagram Protocol through Network Address Translators (STUN) services in that it is a peer-to-peer service. IPOP uses Distributed Hash Tables (DHTs) to store node-specific information. DHTs can provide a key-value lookup service whose data is distributed among the participants in a fault-tolerant way. This allows DHTs to scale to large number of nodes, and handle continuous arrivals and departures of nodes [52].

However, as a result of testing (presented in Section 8.8) it is clear that a general-purpose overlay network based on DHTs will not scale well when a large number of nodes are located at the same physical site. A new model for scheduling across NAT boundaries is needed in that case.

Chapter 7

XMPP and Cloud-based Monitoring

In order to address the problem presented in Section 6.4, a proof-of-concept implementation of a system capable of sending to and receiving information from a VOC is presented. This system utilizes the Extensible Messaging and Presence Protocol (XMPP) in order to traverse NAT. XMPP has been shown to scale to large numbers of nodes behind a single NAT [35]. This chapter discusses XMPP's general use in monitoring applications (Section 7.1) and the utilization of a cloud platform for such a monitoring system (Section 7.2).

7.1 XMPP and Monitoring

XMPP is a powerful way to send messages with minimal overhead. When coupled with a free XMPP provider such as Google Talk, the protocol can be used to send status updates under adverse networking conditions such as Network Address Translation (NAT). Section 7.1.1 gives a brief overview of the protocol, Section 7.1.2 analyzes XMPP messages sent by two different clients, and Section 7.1.3 provides a description of how to send a useful monitoring message with XMPP.

7.1.1 Overview of XMPP

XMPP is a protocol based on the concept of XML streams. Since network communications are generally abstracted as streams of data, viewing XML as a stream instead of a single document is natural. Thus the XML stanza becomes the core element of the protocol. XMPP is based on three core stanzas. The `<message/>` stanza is a "push" mechanism like an email or instant message.

Listing 7.1: XMPP Message stanza generated by Adium

```
<message xmlns='jabber:client' type='chat' id='purplefce9cc26'
         to='michaelfenn87@gmail.com'
         from='cheesy123456789@gmail.com/Macbook3D2CDB2B'>
   <x xmlns='jabber:x:event'>
      <composing/>
   </x>
   <active xmlns='http://jabber.org/protocol/chatstates'/>
   <body>foo</body>
   <html xmlns='http://jabber.org/protocol/xhtml-im'>
      <body xmlns='http://www.w3.org/1999/xhtml'>
         <span style='font-family: Helvetica; font-size: medium;
                      background: #ffffff;'>
            foo
         </span>
      </body>
   </html>
   <nos:x xmlns:nos='google:nosave' value='disabled'/>
   <arc:record xmlns:arc='http://jabber.org/protocol/archive'
               otr='false'/>
</message>
```

The `<presence/>` stanza is a subscription mechanism similar to an RSS feed. Finally, the `<iq/>` stanza is a query-response mechanism. similar to the familiar HTTP request. [34]

Many messaging protocols rely on a single well-known set of core servers in order to facilitate user authentication and initial handshakes. XMPP is a federated system with many, potentially-isolated, authentication domains and namespaces. In essence, anyone can run an XMPP server and have their own set of clients. Federation between XMPP servers is possible, as well as having multiple points-of-presence for a particular identifier, giving rise to the three-part identifier, known as a JID. A typical JID takes the form of `node@domain/resource`. The node part of the JID is a unique identifier within the specified domain. The resource part allows different clients belonging to the same node to be differentiated. The domain part of the JID can also be used to look up a DNS SRV (service) record. The SRV record can define the location and port number of the XMPP server servicing that domain, thus freeing XMPP from the need to have a single root server.

7.1.2 Example XMPP Message Stanzas

Listings 7.1 and 7.2 show two XMPP message stanzas as generated by the Adium [53] IM client and the xmpppy [54] Python library respectively.

Listing 7.2: XMPP Message stanza generated by xmpppy

```
<messages xmlns='jabber:client' to='michaelfenn87@gmail.com' type='chat'
          id='3' from='tigers.cloud@gmail.com/C2EB8B31'>
   <body>foo</body>
   <nos:x xmlns:nos='google:nosave' value='disabled'/>
   <arc:record xmlns:arc='http://jabber.org/protocol/archive' otr='false'/>
</message>
```

The *message* tag itself is given similar attributes by both clients. There is an *xmlns* attribute defining that this element is a part of the *jabber:client* namespace. Each also contains attributes describing who the message is to and from. Notice that the *from* attribute describes the exact originating resource whereas the *to* attribute does not. The XMPP server will decide which resources are to receive this message. The *type* attribute is set to "chat" to indicate that this is a normal chat message. The *id* attribute is an identifier for this particular message from the given JID to the other given JID. Note that this id should be unique; the Adium client does a reasonable job of ensuring that this is the case while the xmpppy library does not. Both clients also include the *body* tag which contains a plain-text message.

The Adium client adds a few additional elements which are not strictly necessary for communication but improve the user's chat experience. The (deprecated) *x* element contains conversation even notifications such as the *composing* tag. This tag allows the receiver's client to display a "User is currently typing..." type of message. The *active* tag supersedes is part of the chat states namespace and supersedes the event notification scheme. In particular the active tag indicates that the user is actively participating in the conversation. The *html* tag provides an HTML version of the sender's message should the receiving client wish to display it in HTML rather than as plain text. As UI experience tags, these are not needed for the internal communications of an XMPP monitoring system, as there is no human to please.

The final two tags, *nos:x* and *arc:record*, are not actually generated by the clients, but are instead added to the messages as they pass though the Google Talk servers. nos:x is a Boolean value which determines whether or not the chat will be archived in the sender's and receiver's Gmail mailboxes. The arc:record accomplishes much the same function, defining this message as a record in the archive, and indicating that it is not "Off-The-Record" (OTR).

Listing 7.3: Constructing an XMPP message with xmpppy

```
import xmpp
jid = xmpp.protocol.JID(conf['userid'])
cli = xmpp.client.Client(jid.getDomain(), debug=[])
cli.connect()
cli.auth(jid.getNode(), conf['password'])
cli.send(xmpp.protocol.Message(target, msg, typ="chat"))
```

7.1.3 Implementing a Monitoring Program with xmpppy

As mentioned above, xmpppy is a Python library for creating XMPP messages. To implement a monitoring program, machine state information is first gathered using standard techniques. Then an XMPP server must be connected and authenticated against, the message constructed, and the message sent to its intended recipient. Listing 7.3 is a listing of example Python code to do this.

First a JID object is constructed from the given user id. The user id is given as *node@domain* so the JID object contains these along with a new resource string appended. Next a Client object that represents an abstract way to perform the various client functions is constructed. The server to connect to is given by the domain contained in the JID. The xmpppy library defaults to always printing debug messages, so its important to set the list of debug events to the empty list for operational purposes. Next the Client object's connect method is called. The connect method uses the DNS SRV record for the domain to determine the exact hostname and port of the server to which it should connect. The auth method authenticates the client with the JID's node name and the node's password. It is at this point that the resource identifier is also bound to this particular Client object. Next, a Message object is constructed giving the recipient, the message body, and the type of the message. The message object is an abstract representation of the actual message element described in Section 7.1.2. Finally, the Message object is sent using the client's send method.

7.2 Google App Engine

Google App Engine (GAE) is a service that opens up Google's infrastructure to anyone who wishes to develop on its platform. GAE allows for transparent scaling of user applications in response to load. GAE also provides a database implementation known as the GAE Datastore. Of particular interest to those developing a monitoring program, GAE allows for a permanent XMPP presence on the Google Talk XMPP network. Section 7.2.1 gives an overview of the GAE service,

Listing 7.4: Handling an XMPP message in GAE

```
class XMPPHandler(webapp.RequestHandler):
    def post(self):
        message = xmpp.Message(self.request.POST)
        logging.debug("xmpp: raw Message sender " + message.sender)
        message.reply("This is a reply.")
```

Section 7.2.2 gives a detailed look at how to interact with the XMPP API presented by GAE, and Section 8.9 discusses some performance implications of GAE's datastore.

7.2.1 Overview of Google App Engine

The GAE platform allows for transparent, dynamic scaling of a web application and is provided on a fee-for-use basis. These fees are implemented by a daily quota system. At the time of this writing there are thirty-four (34) different resource quotas, grouped into nine (9) categories. Each of these quotas represents a resource that may be consumed without charge. Once a quota is exceeded, the application owners are then billed for the overage at a rate determined by the type of resources. The quota categories are: requests, datastore, mail, UrlFetch, Image Manipulation, Memcache, XMPP, Task Queue, and Deployments. GAE also provides an abstraction and API for handling XMPP messages which is discussed below.

The GAE datastore is the method by which GAE applications store persistent data. It is interacted with via a SQL-like language known as GQL. The datastore differs from the databases that many web developers are familiar with in that it is not relational. It is a purely flat database more akin to a set of spreadsheets that can be accessed via a powerful query engine.

7.2.2 Using XMPP with Google App Engine

Listing 7.4 presents a GAE request handler object that interacts with the GAE XMPP API. Since all interactions with a GAE application must adopt the Web-centric request-response paradigm, the receipt of XMPP messages is presented to the application as an HTTP POST to the `/_ah/xmpp/message/chat` URL to which the XMPPHandler class is attached (attachment not shown). A Message object is constructed from the raw XML given in the POST. This message object contains fields corresponding to the sender, the intended recipient, and the body. A reply can be sent by calling the reply method of the message object. This sends a new message back to

the JID given in the sender field. It is clear that the GAE XMPP API is much more limited than the xmpppy version. This makes the API simpler to use, but also limits its functionality. This is perhaps because Google App Engine applications are already tied to Google's ecosystem, and thus communication over any other XMPP network except for Google Talk is seen as unnecessary.

Chapter 8

Results

Building upon previous work [55], four different performance tests were executed to evaluate the VOC: the standard HPL [56] benchmark (Section 8.1), VOC node boot time measurements (Section 8.2), the High Performance Computing Challenge (HPCC) benchmark that complements HPL with measures of bandwidth and additional floating point operations (Section 8.3), Xen host performance (Section 8.4), and HPL block size tuning where sensitivity of the overall performance is measured with respect to various block sizes (Section 8.5). Fundamental performance tests regarding the scalability of IPOP and the performance of the Google App Engine datastore are presented in Sections 8.8 and 8.9, respectively. Dynamic provisioning of VOCs was also tested, with results presented in Section 8.6. Following these tests, a long-term operational test of several VOCs was conducted with results appearing in Section 8.7. Multi-site VOC testing is also presented in Section 8.8. Results presented in this section have been presented in various publications of the author. These publications are cited as appropriate.

8.1 High Performance Linpack (HPL)

The prototype 16 node physical cluster has a theoretical peak of 341 GFLOPS, calculated using 2 CPU cores per node clocked at 2.66 GHz, with 4 FLOPS per cycle per core. HPL tests are performed to compare the actual performance to the theoretical peak. The following HPL parameters are optimized for the prototype cluster and remained constant throughout these tests: block size (NB), process mapping (PMAP), threshold, panel factorization (PFACT), recursive stopping cri-

Table 8.1: Boot Times (seconds)

Physical Node			VM
Statistic	Total Boot	Actual Boot	VM Boot
Minimum	160	43	61.2
Median	160.5	44	65.4
Maximum	163	46	70.2
Average	160.9	44.5	65.5
Std Deviation	1.03	1.09	2.54

terium (NBMIN), panels in recursion (NDIVs), recursive panel factorizations (RFACTs), broadcast (BCAST), lookahead depth (DEPTH), SWAP, swapping threshold, L1 form, U form, Equilibration, and memory alignment. The problem size (N) can be derived with the formula:

$$N = \sqrt{nDU} \tag{8.1}$$

where n is the number of nodes tested, D is the number of double-precision floating-point numbers that can fit into a single node's memory (bytes of node memory / 8), and U is the ratio of memory available for user processes to total memory (U=0.8 leaves space for OS processes). All tests are run with ATLAS 3.8.1 (tuned separately for physical nodes and VOC nodes) and MPICH2 1.0.5p4.

For the physical cluster, the optimal value of N was computed to be 83,000. With this value of the HPL problem size, the performance of the physical hardware was measured at 190 GFLOPS, or 56% of theoretical peak, a reasonable value for a cluster utilizing standard Gigabit Ethernet networking. [9, 42]

8.2 Boot Times

In order to determine the practicality of scheduling VMs as processes, boot times were measured and compared to the physical hardware. An XML-RPC boot timing server is deployed to monitor the virtual systems. Boot times for the physical nodes are measured by hand with a chronograph. Table 8.1 summarizes boot time test results.

The physical boot process is divided into three phases: Pre-eXecution Environment (PXE) timeout, a GRand Unified Bootloader (GRUB) timeout, and the actual kernel boot time. The Actual Boot column does not include either the PXE or the GRUB timeouts. The VM boot time

measures the amount of time from KVM initiation on the host until all daemons had been started on the guest. Approximately 10 more processes were found to be running on the VOC node than on the physical host. On average, the VOC nodes require an extra 21 seconds to boot. [9, 42]

8.3 High Performance Computing Challenge Benchmark

The High Performance Computing Challenge (HPCC) [57] benchmark suite is a collection of well-known HPC benchmarks, packaged together in a convenient format. The benchmark consists of seven tests which produce nine data points. A short description of each test follows:

- **HPL,** measuring floating point performance for matrix multiplication in GFLOPS and is discussed in more detail in Section 8.1 above,
- **PTRANS,** measuring overall network communications in GB/s,
- **RandomAccess,** measuring integer memory accesses and updates in GUP/s,
- **FFT,** measuring double-precision floating point performance for Discrete Fourier Transforms in GFLOPS,
- **STREAM,** measuring memory bandwidth GB/s,
- **DGEMM,** measuring double-precision floating point performance for matrix multiplication in GFLOPS,
- **b_eff,** measuring bandwidth in GB/s and latency in μs of multiple simultaneous network communications.

GFLOPS measures rate of execution, one GFLOPS is a billion (10^9) floating point operations per second, generally a 64-bit addition or multiplication. GB/s is a measure of throughput (rate of transfer), one GB/s is one billion (10^9) bytes of data transferred per second. GUP/s is a measure of memory update speed, that is, a read from a random address followed by a write to a random address. One GUP/s is one billion (10^9) of these operations in a second. Finally, μs are microseconds, 10^{-6} seconds.

Tables 8.2, 8.3, and 8.4 are comparisons of the VOC to the physical cluster by means of the HPCC benchmark. Tests were run on a single physical node (single process) versus a single VOC

Table 8.2: Physical vs. Virtualized, Single Process

Process Grid	Xen Overhead	KVM Overhead
Problem Size	0%	0%
G-HPL (GFLOPS)	6.566%	8.771%
G-PTRANS (GB/s)	19.415%	12.946%
G-Random Access (GUP/s)	35.519%	15.818%
G-FFTE (GFLOPS)	17.733%	42.370%
EP-STREAM Sys (GB/s)	12.704%	1.491%
EP-STREAM Triad (GB/s)	12.704%	1.491%
EP-DGEMM (GFLOPS)	7.892%	7.977%
RandomRing Bandwidth (GB/s)	N/A	N/A
RandomRing Latency (μs)	N/A	N/A

node. Two tests were then run on the full cluster, the first utilizing one dual-CPU VOC node per physical node and the second with two single-CPU VOC node per physical node. All parameters except problem size (N) and block size (NB) are maintained from the previous HPL tests. Problem sizes are scaled to fit into available VOC memory (1 GiB per CPU per node, see Equation 8.2) according to Equation 8.1

$$Mem_{VOC} = 1GiB \cdot N_{CPU} \cdot N_{VCN} \tag{8.2}$$

Block sizes are increased in order to compensate for latency. The overhead due to virtualization was calculated with the formula:

$$Overhead = \frac{Physical - VOC}{Physical} \cdot 100\% \tag{8.3}$$

In cases when larger values indicate worse performance (i.e. latency) the result of Equation 8.3 is multiplied by -1. Otherwise, negative values indicate increased performance of the VOC relative to the physical cluster.

Under KVM, single-node virtualization overhead ranges from under 10% to around 16% with G-FFTE being an outlier at 42% (Table 8.2). Xen fares similarly, except its outlier is G-Random Access at 35%. However, the full cluster overhead for MPI applications under KVM (Tables 8.3 and 8.4) is quite high at 52% for G-HPL with single-CPU VMs and 85% with dual-CPU VMs. Xen performs much better with penalties of 23% for G-HPL with single-CPU VMs and 30% with dual-CPU VMs. RandomRing latency is approximately three times worse with KVM than with the physical hardware. Associated HPL performance of the VOC was thus quite poor. Also note that Xen's RandomRing latency is about 30% lower than KVM's latency in the same benchmark. This

Table 8.3: Physical vs. VOC, One 2-CPU VM per Physical Node (32 processes)

Process Grid	Xen Overhead	KVM Overhead
Problem Size	0%	0%
G-HPL (GFLOPS)	30.470%	85.173%
G-PTRANS (GB/s)	42.818%	91.985%
G-Random Access (GUP/s)	35.910%	73.082%
G-FFTE (GFLOPS)	24.899%	82.556%
EP-STREAM Sys (GB/s)	-6.151%	-39.889%
EP-STREAM Triad (GB/s)	-6.151%	-39.889%
EP-DGEMM (GFLOPS)	7.269%	16.559%
RandomRing Bandwidth (GB/s)	23.425%	67.419%
RandomRing Latency (μs)	102.611%	290.179%

Table 8.4: Physical vs. VOC, Two VMs per Physical Node (32 processes)

Process Grid	Xen Overhead	KVM Overhead
Problem Size	0%	0%
G-HPL (GFLOPS)	22.935%	52.063%
G-PTRANS (GB/s)	4.302%	44.968%
G-Random Access (GUP/s)	22.941%	70.643%
G-FFTE (GFLOPS)	67.380%	23.449%
EP-STREAM Sys (GB/s)	-5.650%	-23.818%
EP-STREAM Triad (GB/s)	-5.650%	-23.818%
EP-DGEMM (GFLOPS)	6.588%	13.979%
RandomRing Bandwidth (GB/s)	68.779%	-17.148%
RandomRing Latency (μs)	67.259%	206.787%

Table 8.5: Xen vs. non-Xen Kernels, Single Process

Process Grid	1x1 non-Xen	1x1 Xen	Xen Kernel Overhead
Problem Size	10300	10300	0%
G-HPL (GFLOPS)	7.913	7.908	0.063%
G-PTRANS (GB/s)	0.729	0.747	-2.369%
G-Random Access (GUP/s)	0.002	0.001	54.354%
G-FFTE (GFLOPS)	0.799	0.657	17.806%
EP-STREAM Sys (GB/s)	3.866	3.157	18.332%
EP-STREAM Triad (GB/s)	3.866	3.157	18.332%
EP-DGEMM (GFLOPS)	8.348	8.370	0.261%
RandomRing Bandwidth (GB/s)	N/A	N/A	N/A
RandomRing Latency (μs)	N/A	N/A	N/A

is believed to be a contributing factor to KVM's poor HPL performance due to excessive context switching between user and kernel mode in its network code.

The large difference in best-case RandomRing latencies (Table 8.4) between KVM (228μs) and Xen (74μs) can be attributed to Xen's paravirtualization of guest network devices. Xen's network device drivers can make a call directly into the Xen hypervisor, avoiding any context switches. KVM's network device drivers must make a system call, which which is the trapped by the VT instructions and then passed to the user-mode KVM process. This unnecessary context switch would cause additional latency, but it is also possible that Xen's network code is simply more mature than KVM's code. Nevertheless, context switches must play some role in Xen's improved network latencies, and therefore, its improved HPL performance. [9, 42]

Due to formatting constraints, the raw benchmark numbers are omitted from this section. Please refer to Appendix A for a complete listing.

8.4 Xen Domain0 Performance

As mentioned in Section 6.1.2, the Xen hypervisor requires that the host operating system run a modified version of the Linux kernel. In order to determine whether or not the additional logic introduced by the Xen hypervisor code introduces a performance penalty, benchmark measurements were taken comparing the physical cluster running under the normal Linux kernel (non-Xen) and the Xen kernel. The results in Tables 8.5 and 8.6 represent comparisons between the two *physical* cluster kernels. *No* virtual machines are running in either case.

For the full cluster, G-Random Access and RandomRing Latency are the benchmarks most

Table 8.6: Xen vs. non-Xen Kernels, Two Processes per Physical Node (32 processes)

Process Grid	8x4 non-Xen	8x4 Xen	Xen Kernel Overhead
Problem Size	58600	58600	0%
G-HPL (GFLOPS)	169.807	167.693	1.245%
G-PTRANS (GB/s)	0.867	0.853	1.576%
G-Random Access (GUP/s)	0.014	0.003	38.203%
G-FFTE (GFLOPS)	2.287	2.346	-2.594%
EP-STREAM Sys (GB/s)	59.046	59.404	-0.605%
EP-STREAM Triad (GB/s)	1.845	1.856	-0.605%
EP-DGEMM (GFLOPS)	8.271	8.280	-0.116%
RandomRing Bandwidth (GB/s)	0.023	0.023	-3.537%
RandomRing Latency (μs)	74.444	108.993	46.410%

affected by the switch to the Xen kernel, with penalties of 38% and and 46% respectively. This is consistent with the intuitive observation that the Xen hypervisor layer introduces an additional measure of latency. Thus it can be postulated that a Xen physical host's latency would be modeled by Equation 8.4, while a KVM host's latency would be better represented by Equation 8.5, where L_{PHY} is the latency of the network's physical and datalink layers, L_{HYP} is the latency of the Xen hypervisor layer, and L_{OS} is the latency of the OS's network stack implementation. Note that these equations only apply to the *host* OS. Additional terms would be needed to correctly model the latency of the guest OS.

$$L = L_{PHY} + L_{HYP} + L_{OS} \tag{8.4}$$

$$L = L_{PHY} + L_{OS} \tag{8.5}$$

Thus, by merely running a Xen-enabled kernel, a system will incur a performance penalty, even when no virtual machines are running. It is, however, important to compare Tables 8.4 and 8.6 which represent a virtualized cluster and a non-virtualized cluster, respectively. The *additional* overhead imposed by Xen's virtualization is only about 20% more than the overhead of the Xen hypervisor layer. This is also consistent with the Xen model in which the "physical" OS is simply a more privileged VM. [9]

8.5 Block Size (NB) Tuning

To test the hypothesis that increased latency significantly decreases performance for MPI jobs, a test was run with varying block sizes. Figure 8.1 summarizes the results on both the physical cluster and the VOC. HPL uses a tile-based algorithm, dividing the problem matrix into blocks which are then distributed to each worker process. The total number of blocks that need to be distributed is the problem size divided by the block size, so greater block sizes should reduce the total number of transfers, thus reducing the effects of latency. HPL performance increased with increasing block size, reaching an optimal point at a block size of 400. Above this threshold, performance began to decrease as load-balancing became inefficient.

Note that the optimum NB for the physical nodes is very close to that of the VOC, but the maximum of the VOC's plot is much more acute than that of the physical cluster. The function of NB to performance also appears to be more complex than that of the physical node, as indicated by the local maximum at an NB of 250 in the VOC's plot. The implications of this observation are not certain, although the slope of the VOC graph is certain much steeper, so this additional factor could simply be hidden in the physical cluster's plot. [9, 42]

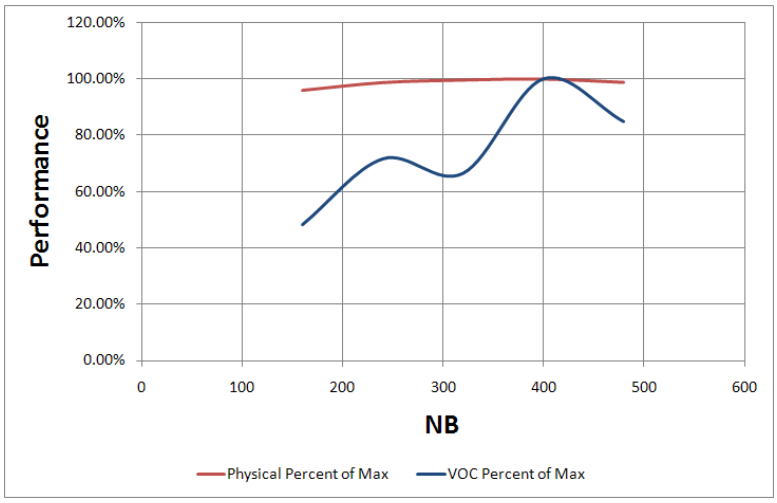

Figure 8.1: NB vs. Performance

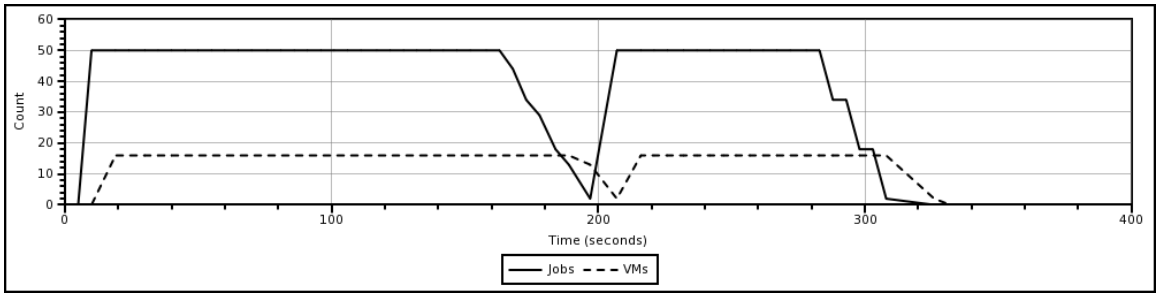

Figure 8.2: Two submissions of 50 jobs, 10-second execution time, submitted locally

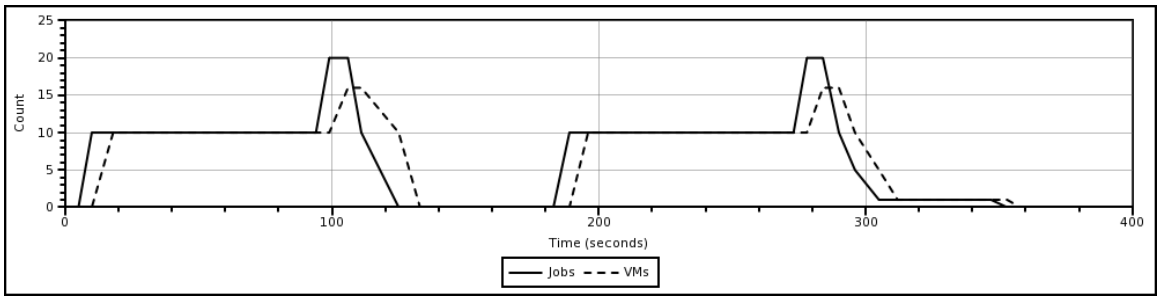

Figure 8.3: Submitted 10 jobs every 90 seconds, 10-second execution time, submitted locally

8.6 Dynamic Provisioning of Virtual Organization Clusters

Since the basic viability of virtualization for scientific computing is established, a prototype Virtual Organization Cluster was implemented, and tests of the system are presented. These tests consist of several micro-benchmarks:

- Two submissions of 50 jobs each in order to simulate the arrival of two, overlapping batches of jobs.
- Sets of 10 jobs, each with a 10-second execution time, submitted 90 seconds apart
- Sets of 10 jobs, each with a 10-second execution time, submitted 30 seconds apart
- Sets of 10 jobs, each with a 1-second execution time, submitted 30 seconds apart

These tests allow observation of the behavior of the system under regular periodic loads. These types of loads fairly approximate the loads encountered on the Open Science Grid, albeit compressed temporally. [51]

Figure 8.2 shows that the VOC watchdog had started the maximum number of VMs by 20 seconds into the test. The number of VMs remained at at the maximum until the first batch of jobs

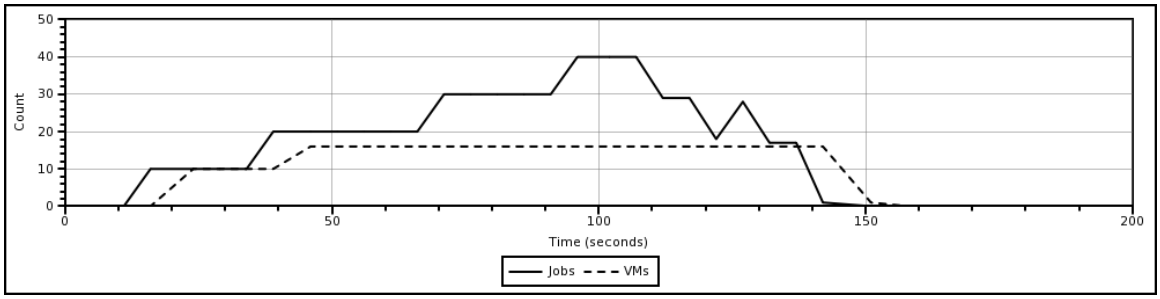

Figure 8.4: Submitted 10 jobs every 30 seconds, 10-second execution time, submitted locally

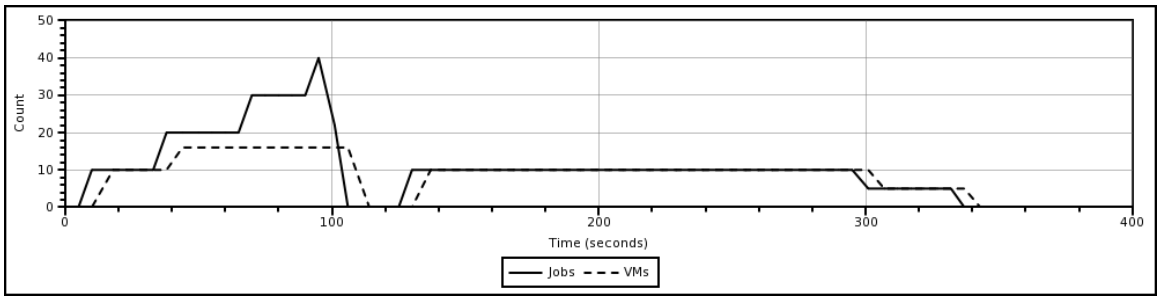

Figure 8.5: Submitted 10 jobs every 30 seconds, 1-second execution time, submitted locally

began completing, at which time the watchdog began stopping VMs. However, this figure shows that the watchdog exhibited over-responsive behavior, as it was attempting to stop VMs even as the second burst of jobs began entering the queue.

Figures 8.3 through 8.5 show that the watchdog exhibits predictable behavior with regard to the queue size. The lag between a job entering the queue and a VM being started can be attributed the periodic nature of the watchdog as well as to VM boot times. The lag between the job completing and the VMs being stopped can be attributed to the periodic nature of the watchdog.

8.7 Operational VOC Testing

Operational tests performed to confirm the validity of the previously conducted synthetic testing is presented in this section. Section 8.7.1 presents the results of testing a VSP-supplied VM image with the Engage and NanoHUB OSG VOs while Section 8.7.2 presents the results of testing with an VM image supplied by the STAR VO.

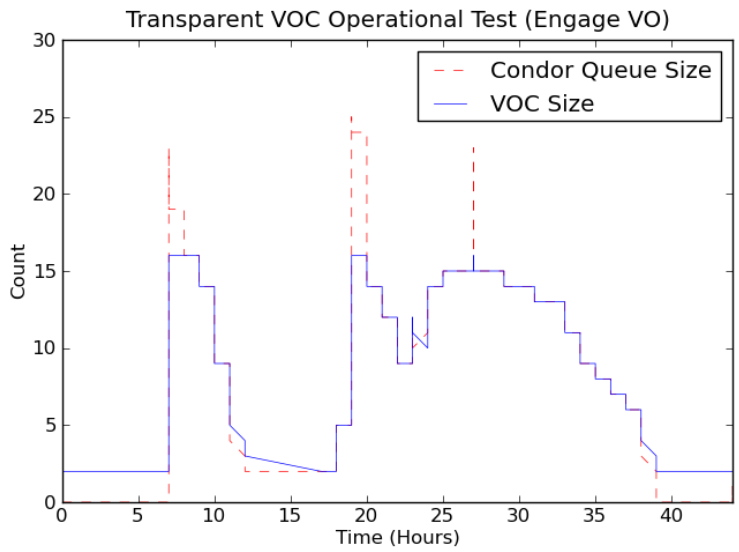

Figure 8.6: Short operational test (44 hours) with the physical cluster configured to support a 16-node Virtual Organization Cluster dedicated to the Engage Virtual Organization.

8.7.1 Engage and NanoHUB VO Testing

Once synthetic tests determined the viability of the VOC prototype, it was made operational on the Open Science Grid. The maximum VOC size was set at 16 nodes, the minimum VOC size was set at 2 nodes, and VOCs were defined for the Engage and NanoHUB VOs. Two operational tests were performed, a short operational test and a long operational test. Each VOC is comprised of 32-bit CentOS VMs.

On June 1, 2009, the short test started, with only the Engage VOC active. After approximately 44 hours of testing, the VOC was removed from service. Figure 8.6 illustrates the results of this test. Several bursts of jobs arrived, each of which caused the VOC to expand then contract. The VOC size reached the maximum (16 nodes) on two occasions.

On June 4, 2009, a longer operational test was begun. This test used the same operational parameters as the earlier operational test. However, the NanoHUB VOC was allowed to run concurrently with the Engage VOC during this test. The long operational test ended on August 17, 2009. During the long operational test, jobs arrived in bursts from the Engage VO (Figure 8.7) during the the first 1100 hours of testing. The number of jobs arriving after that time was low. Figure 8.8 shows a similar pattern, except that the jobs only arrive before the 650th hour of testing. The lack of jobs arriving during the later months of the tests can be partially attributed to the annual reduction of

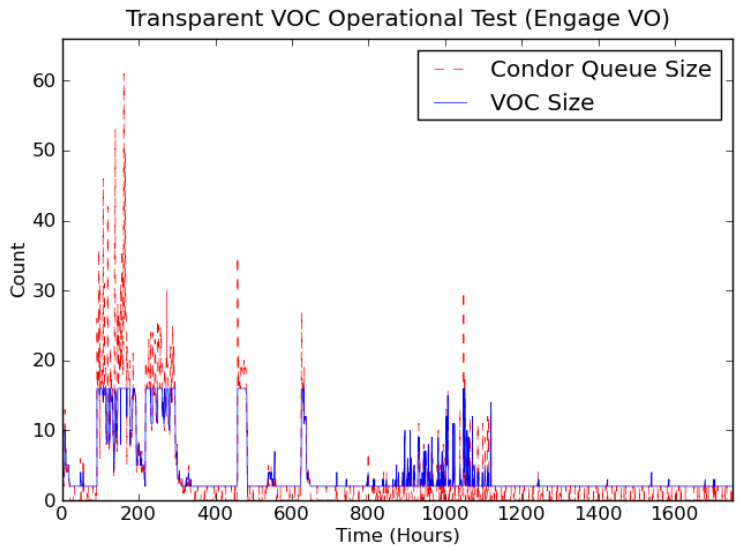

Figure 8.7: Long operational test: Engage VO. A second operational VOC, dedicated to the NanoHUB VO, was sharing the same hardware.

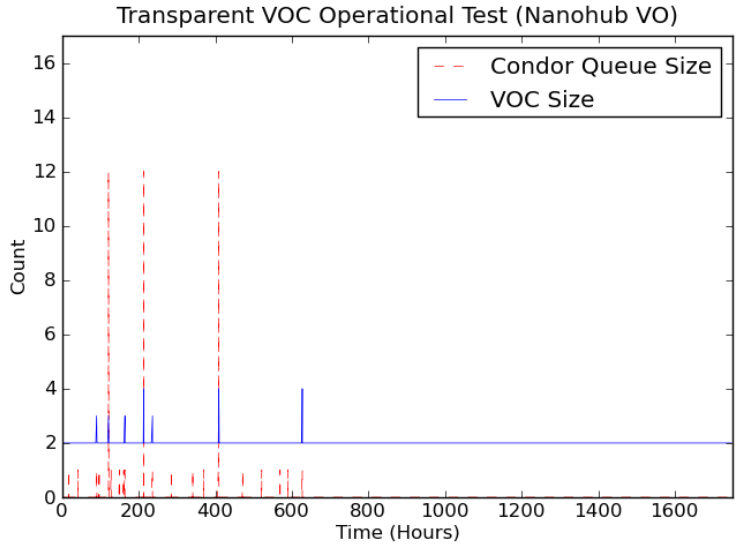

Figure 8.8: Operational test: NanoHUB VO. A second operational VOC, dedicated to the Engage VO, was sharing the same hardware.

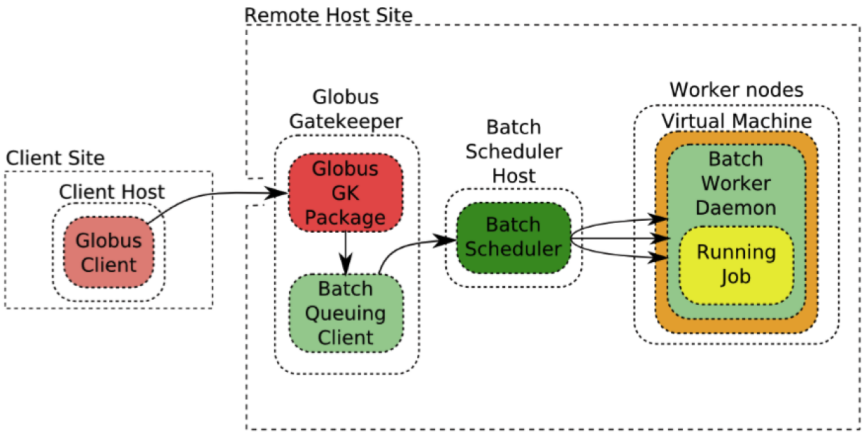

Figure 8.9: Integration of the STAR VM into the prototype VOC

the number of jobs on OSG during the summer months. The large-scale migration away from 32-bit architecture jobs during this time-frame can also explain the lack of jobs as August approached.

8.7.2 STAR VO Testing

The STAR VO provided an image that was contextualized for the prototype cluster using the procedures outlined in Section 6.1.5. Figure 8.9 depicts the STAR VM's integration with the prototype VOC. Once the VM has image-level contextualization performed, it appeared to the STAR VO in the same manner as any of their other resources. These results use the same watchdog parameters as the previous Engage and NanoHUB operational testing.

STAR utilized the 16 VMs available and submitted 32 jobs. The jobs 280MB of total output which was streamed back to the Brookhaven National Laboratory (BNL) at 6.8MB/s. The total processing time was approximately 11 hours and 7 minutes of VOC boot latency were observed. The boot time plus virtualization overhead combined to give total VOC overhead of approximately one percent over a local test by BNL.

As shown in Figure 8.10, the VOC's Condor scheduler was fast for the first two jobs due to the fact that the watchdog was configured to keep two VMs running at all times. Jobs 2 through 16 started as soon as a VM was started and joined to the Condor pool. Jobs 17-32 were forced to wait in the queue because there were only 16 VOC nodes available. Once the first 16 jobs completed, Condor was able to schedule the remaining 16 jobs to the VOC nodes without delay. [48]

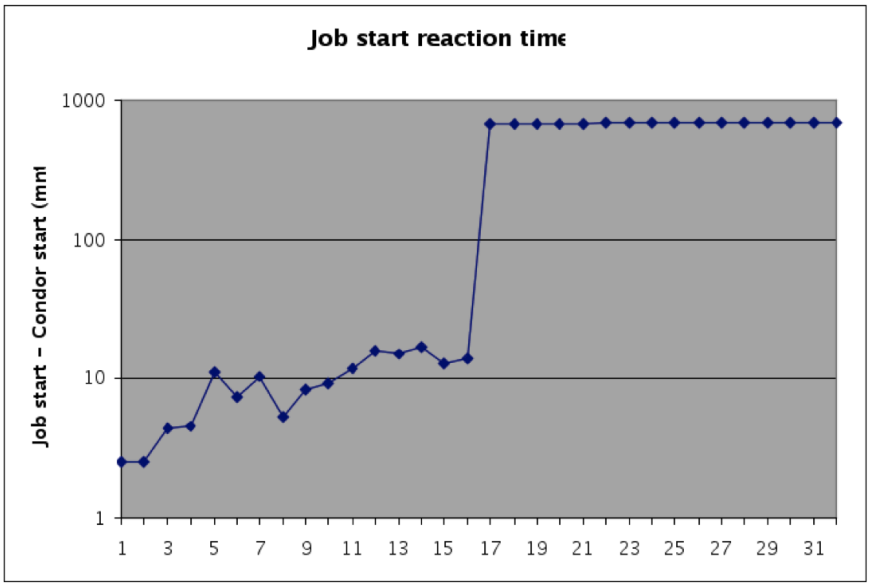

Figure 8.10: Condor reaction time as observed by STAR

8.8 Multi-site VOC Testing

As a VOC should ideally be able to include worker nodes from multiple-sites, a test was conducted to determine the scalability limits of IPOP. This test was run on the Clemson University Palmetto cluster because the prototype cluster is not large enough to test the scalability limits of IPOP. In order to effect this test, 200 simultaneous requests for eight VMs were made to Clemson University's Palmetto Cluster. One large request for 1600 VMs would not be able to be serviced by the cluster due to other load on the system. One IPOP node was instantiated on each VM.

Figure 8.11 shows the number of IPOP nodes at each point in time, as well as the cumulative total number of unique nodes observed. IPOP is unable to scale beyond 500 nodes at any given point in time. This is likely due to the peer-to-peer overlay becoming unbalanced because of a large influx of nodes behind a small number of NAT gateways. This influx is problematic because it increases the relative loads on each system not behind that set of NAT gateways, ultimately leading to a collapse of the overlay. This is further evidenced by the fact that 18.75% of VMs were never able to properly instantiate their IPOP instances.

Once the IPOP scaling limit was determined, a test could be conducted using the local prototype VOC implementation. This test involved provisioning VMs running on the local cluster alongside VMs running in Amazon's Elastic Compute Cloud (EC2). The watchdog was modified to

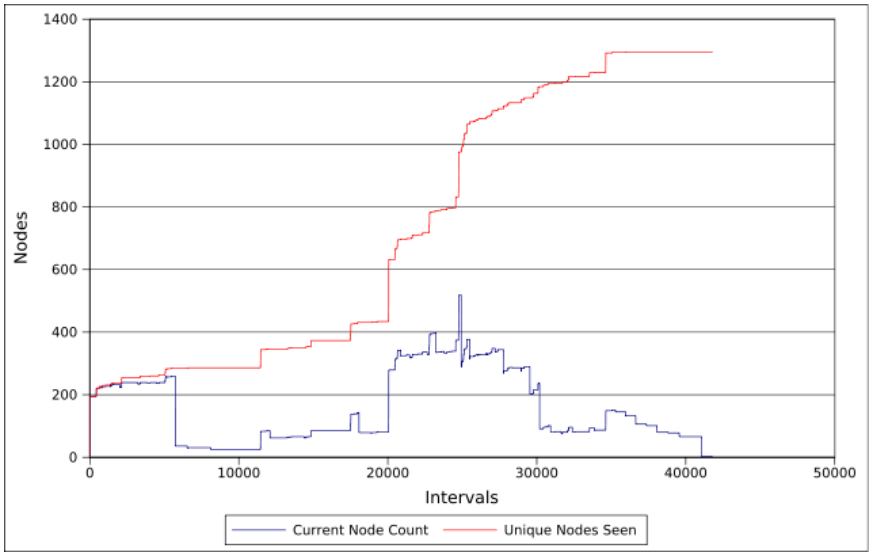

Figure 8.11: IPOP Scaling

allow VMs in excess of the local site maximum (16 in this case) to be started on EC2. These VMs used IPOP to join the same scheduling pool as the local VMs. As EC2 VMs have a higher monetary cost than local VMs, the watchdog assigned the EC2 VMs low priority for starting and high priority for stopping. A limit of 100 simultaneously-running EC2 VMs is also imposed.

Figure 8.12 illustrates the results of a test where 300 jobs were submitted, each with an execution time of 10 minutes. The watchdog performs as expected, starting local VMs up to the local maximum (16) and then starting EC2 VMs up to the EC2 maximum (100). As the size of the queue dropped below 116, the EC2 VMs were stopped first, followed by the local VMs. Even though the local VMs were behind a NAT boundary, and the EC2 VMs were geographically distant from the local cluster, scheduling performance was not adversely affected.

8.9 Google App Engine Datastore Performance

When implementing the front-end of the monitoring system in GAE, it was noticed that some pages were taking a noticeably longer time to load than others. This was traced to the Datastore API calls and a performance investigation was begun.

The test consisted of generating random, realistic datastore records consisting of two integer fields, two string fields, and a date field. 250 of these records were generated and then inserted into the datastore. Then all 250 records were selected with a GQL query and a field from the each record

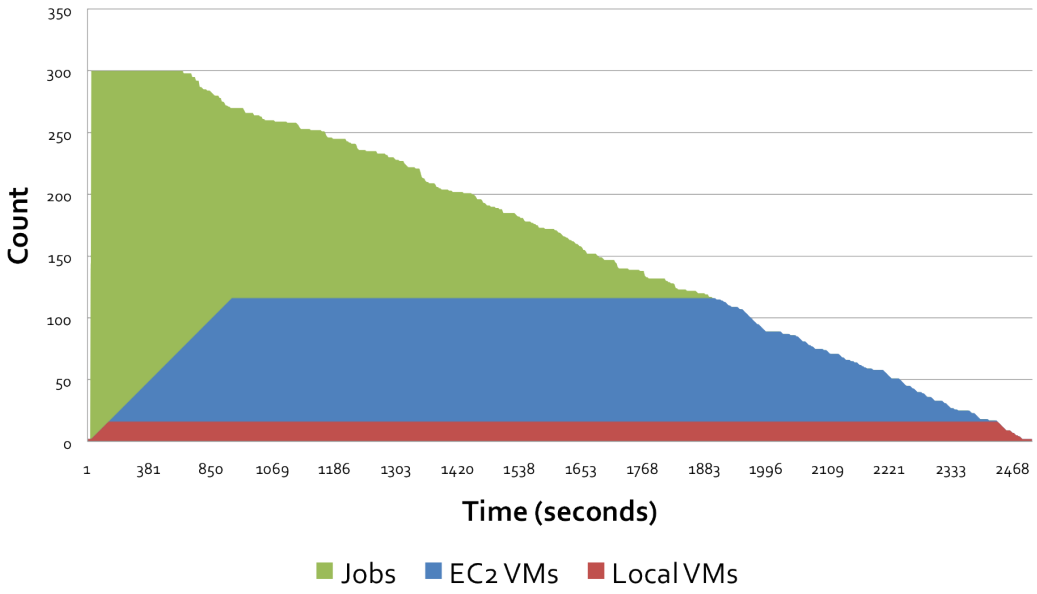

Figure 8.12: 300 10-minute jobs submitted to a multi-site VOC

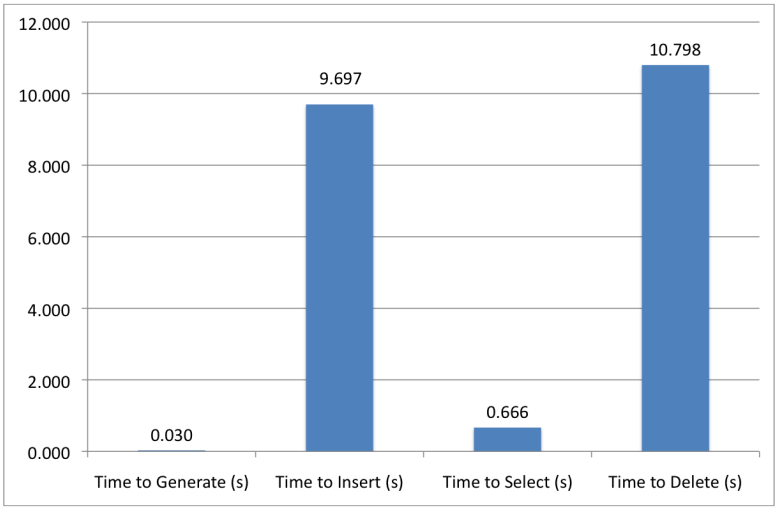

Figure 8.13: Datastore performance on GAE

is displayed. Finally, the records were deleted. This test was run five times on GAE itself and five times in the local test environment provided by the GAE SDK.

Figures 8.13 and 8.14 present the average run times of the various stages of the test on both GAE and the local testbed. Since all records are generated before any record is inserted, the time taken to generate the records is a good indicator of relative CPU performance. These times were similar on both GAE and the local testbed, leading to the conclusion that Google's servers have about the same CPU power as a 2.4GHz Core 2 Duo processor when executing a single-threaded process. Thus, the poor performance exhibited in the other phases is probably not indicative of a lack of CPU power.

The GAE platform is relatively fast at selecting records from the datastore. The local datastore was only about 159% faster than GAE. This is not a bad result considering that network communication time is essentially nil on the local testbed since all components are running on the same machine.

The GAE platform exhibits poor insert and delete performance. It takes about 10 seconds to insert or delete the 250 records, or 0.04 seconds per record. For comparison, the local testbed is able to insert or delete 166 records in the same time it takes the GAE platform to insert or delete a single record.

This result has serious performance implications for GAE applications. Care must be taken

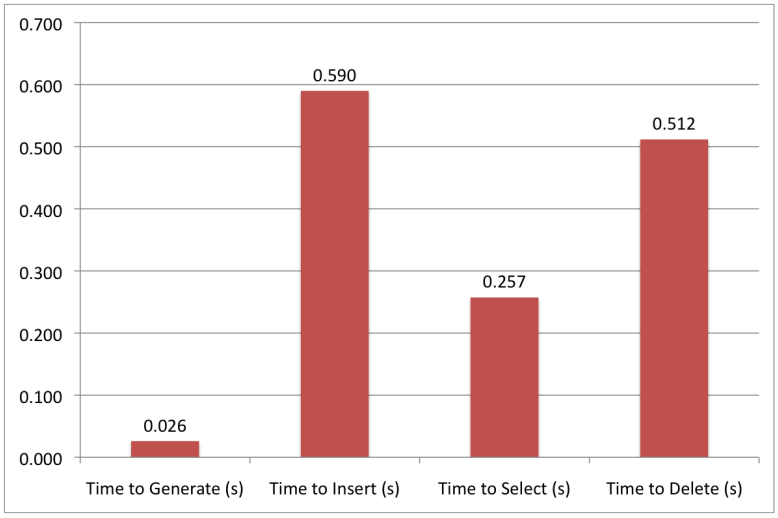

Figure 8.14: Datastore performance on local testbed

to avoid inserting or deleting too many records at once, especially since a GAE request is on a hard execution deadline of about 25 seconds.

Chapter 9

Conclusions

Based on the preliminary results of the study, it can be concluded that KVM is generally efficient when network I/O latency is not a factor, as demonstrated by the low single node overhead in HPL. Some unusual results for the STREAM benchmark on the full cluster were encountered. This benchmark exhibits low temporal locality and high spatial locality according to Luszczek *et. al.* [57]. An investigation is ongoing with regard to why virtualization would improve performance in these situations. [9, 42]

As shown in Tables 8.3 and 8.4, network latency is poor (there is a three-fold increase), resulting in large virtualization overhead. Based on prior MPI studies and our own HPL testing, the high latency of the virtual network causes the poor HPL performance [58, 57]. However, HPL provides a good diagnostic tool because it distributes blocks of a size specified by the NB parameter, allowing the nature of its network traffic to be controlled to some degree. As Figure 8.1 shows, HPL performs better under high-latency conditions when the block size is increased. This performance improvement is due to the fact that fewer, larger transfers will be less affected by latency than many small transfers. HPL performance eventually drops off due to poor load balancing with greater block sizes. Latency reduction is a crucial need to make virtual clusters a mainstream HPC technique. [9, 42]

While the loss in performance of inter-node communication with MPI is disappointing, these types of jobs are not common on the Open Science Grid. Therefore, KVM does appear well-suited to Condor jobs in the vanilla and standard universes, which OSG sites primarily utilize [59]. 8.7%

is believed to be an acceptable performance overhead in these situations, given the benefits gained in terms of VO compute environment customization. [9, 42]

As an interesting aside, results show that Xen-enabled kernels should not be run on machines which are not intended to be hypervisor providers. Such a configuration creates a situation where performance penalties, nearly 20% in some cases, are incurred for no benefit. This finding is in line with the common wisdom that holds that one should not enable any unnecessary services; unfortunately, many Linux distributions will enable a Xen kernel by default. [9, 42]

KVM is a promising virtual machine monitor for grid computing. It is easily deployed (compared to Xen) and is simple to maintain while still providing good performance for many Condor jobs. While there are some issues with virtual networking, the results show that KVM is a viable VMM for Open Science Grid sites. [9]

The fundamental viability of virtualization for high-throughput computing allows further work on the VOC Model to remain relevant. To that end, the prototype VOC was extended to provide dynamically-provisioned and multi-site capabilities. Testing showed that these capabilities performed as expected as long as the total number of VMs did not exceed the IPOP scaling threshold of approximately 500 machines. Above this threshold, a different approach to scheduling must be taken.

The combination of XMPP and a cloud-based platform such as Google App Engine allows for the robust receipt, aggregation, and retrieval of virtual machine status information. The use of XMPP allows the VMs to be behind NAT networking and still give useful data. The use of a stable, easily-reached cloud endpoint greatly simplifies the overall system and allows a central hub for information to be present. The cloud-based nature of this endpoint means that it is not subject to the single-point-of-failure and scaling concerns that a traditional well-known endpoint (i.e. a server) would be.

The performance of the GAE datastore is disappointing given that a monitoring application needs to constantly update its records. Clever optimization techniques can be employed to reduce the number of insertions and deletions that are necessary, but can never totally remove this need. Perhaps as more data-intensive applications move to the cloud, Google will improve the performance of this aspect of the GAE service.

A cloud-based monitoring system utilizing XMPP was shown to be viable. Current work on

Kestrel [35], shows that an XMPP-based scheduler can scale far beyond the limits of IPOP. Current limitations of the Google App Engine API may preclude the implementation of Kestrel in the cloud.

Thus, this work has shown that the theoretical Virtual Organization Cluster Model is practical to implement. A working prototype was created which was able to explore the scalability limitations of current overlay networking technologies. The VOC Model does not require an overlay network, only a scheduler which is capable of traversing NAT boundaries. Therefore, the VOC Model itself is not constrained by these technologies, it simply requires a new approach to scheduling jobs over a wide area.

Appendices

Appendix A Full Benchmarking Results

Any benchmarks results which could not fit into the results section due to formatting constraints appear below.

Table A.1: Physical vs. Virtualized, Single Process

Process Grid	1x1 Physical	1x1 Xen VOC	Xen Overhead	1x1 KVM VOC	KVM Overhead
Problem Size	10300	10300	0%	10300	0%
G-HPL (GFLOPS)	7.913	7.393	6.566%	7.218	8.771%
G-PTRANS (GB/s)	0.729	0.588	19.415%	0.635	12.946%
G-Random Access (GUP/s)	0.002	0.001	35.519%	0.002	15.818%
G-FFTE (GFLOPS)	0.799	0.658	17.733%	0.461	42.370%
EP-STREAM Sys (GB/s)	3.866	3.375	12.704%	3.808	1.491%
EP-STREAM Triad (GB/s)	3.866	3.375	12.704%	3.808	1.491%
EP-DGEMM (GFLOPS)	8.348	7.689	7.892%	7.682	7.977%
RandomRing Bandwidth (GB/s)	N/A	N/A	N/A	N/A	N/A
RandomRing Latency (μs)	N/A	N/A	N/A	N/A	N/A

Table A.2: Physical vs. VOC, One 2-CPU VM per Physical Node (32 processes)

Process Grid	8x4 Physical	8x4 Xen VOC	Xen Overhead	7x4 KVM VOC	KVM Overhead
Problem Size	58600	58600	0%	58600	0%
G-HPL (GFLOPS)	169.807	118.067	30.470%	25.178	85.173%
G-PTRANS (GB/s)	0.867	0.496	42.818%	0.069	91.985%
G-Random Access (GUP/s)	0.014	0.009	35.910%	0.004	73.082%
G-FFTE (GFLOPS)	2.287	1.717	24.899%	0.399	82.556%
EP-STREAM Sys (GB/s)	59.046	62.678	-6.151%	82.599	-39.889%
EP-STREAM Triad (GB/s)	1.845	1.959	-6.151%	2.581	-39.889%
EP-DGEMM (GFLOPS)	8.271	7.669	7.269%	6.901	16.559%
RandomRing Bandwidth (GB/s)	0.023	0.017	23.425%	0.007	67.419%
RandomRing Latency (μs)	74.444	150.831	102.611%	290.463	290.179%

Table A.3: Physical vs. VOC, Two VMs per Physical Node (32 processes)

Process Grid	8x4 Physical	8x4 Xen VOC	Xen Overhead	7x4 KVM VOC	KVM Overhead
Problem Size	58600	58600	0%	58600	0%
G-HPL (GFLOPS)	169.807	130.862	22.935%	81.401	52.063%
G-PTRANS (GB/s)	0.867	0.830	4.302%	0.447	44.968%
G-Random Access (GUP/s)	0.014	0.011	22.941%	0.004	70.643%
G-FFTE (GFLOPS)	2.287	0.746	67.380%	1.751	23.449%
EP-STREAM Sys (GB/s)	59.046	62.382	-5.650%	73.110	-23.818%
EP-STREAM Triad (GB/s)	1.845	1.949	-5.650%	2.285	-23.818%
EP-DGEMM (GFLOPS)	8.271	7.726	6.588%	7.114	13.979%
RandomRing Bandwidth (GB/s)	0.023	0.007	68.779%	0.027	-17.148%
RandomRing Latency (μs)	74.444	125.258	67.259%	228.383	206.787%

Appendix B Full HPCC Parameters

Listing B.1: hpccinf.txt

```
HPLinpack benchmark input file Innovative Computing Laboratory,
    University of Tennessee
HPL.out       output file name (if any)
6             device out (6=stdout,7=stderr,file)
1             # of problems sizes (N)
83000         Ns
1             # of NBs
400           NBs
0             PMAP process mapping (0=Row-,1=Column-major)
1             # of process grids (P x Q)
8             Ps
4             Qs
16.0          threshold
1             # of panel fact
1             PFACTs (0=left, 1=Crout, 2=Right)
1             # of recursive stopping criterium
8             NBMINs (>= 1)
1             # of panels in recursion
2             NDIVs
1             # of recursive panel fact.
2             RFACTs (0=left, 1=Crout, 2=Right)
1             # of broadcast
1             BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1             # of lookahead depth
1             DEPTHs (>=0)
2             SWAP (0=bin-exch,1=long,2=mix)
64            swapping threshold
0             L1 in (0=transposed,1=no-transposed) form
```

```
0              U  in (0=transposed,1=no-transposed) form
1              Equilibration (0=no,1=yes)
8              memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator). ######
0                        Number of additional problem sizes for PTRANS
1200 10000 30000                 values of N
0                                number of additional blocking sizes for
    PTRANS
40 9 8 13 13 20 16 32 64     values of NB
```

Appendix C Self-citation Policy

This work contains many sections previously published by the author. The author followed the Association for Computing Machinery's (ACM) policy [60] regarding self-citation. That policy is reproduced below:

> "Self-plagiarism is a related issue. In this document we define self-plagiarism as the verbatim or near-verbatim reuse of significant portions of one's own copyrighted work without citing the original source[2]. Note that self-plagiarism does not apply to publications based on the author's own previously copyrighted work (e.g., appearing in a conference proceedings) where an explicit reference is made to the prior publication[3]. Such reuse does not require quotation marks to delineate the reused text but does require that the source be cited.
>
> "[2] See Collberg and Kobourov, `http://portal.acm.org/citation.cfm?doid=1053291.1053293`.
>
> "[3] Manuscripts submitted to ACM Journals and Transactions based on the author's own previously copyrighted work (e.g., appearing in a conference proceedings) must be disclosed at the time of submission and an explicit reference to the prior publication must be included in the submitted manuscript. The norm for ACM Journals and Transactions is that the submitted manuscript must contain at least 25% new content material (i.e., material that offers new insights, new results, etc.). For more details see `http://www.acm.org/pubs/sim_submissions.html` ."

Bibliography

[1] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," *International Journal of Supercomputing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[2] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, "A case for grid computing on virtual machines," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[3] B. Quetier, V. Neri, and F. Cappello, "Selecting a virtualization system for Grid/P2P large scale emulation," in *Proceedings of the Workshop on Experimental Grid Testbeds for the Assessment of Large-scale Distributed Applications and Tools (EXPGRID'06)*, Paris, France, June 2006.

[4] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, "From virtualized resources to virtual computing grids: the In-VIGO system," *Future Generation Computer Systems*, vol. 21, no. 6, pp. 896–909, June 2005.

[5] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, "VMPlants: Providing and managing virtual machine execution environments for grid computing," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004.

[6] A. Matsunaga, M. Tsugawa, M. Zhao, L. Zhu, V. Sanjeepan, S. Adabala, R. Figueiredo, H. Lam, and J. A. Fortes, "On the use of virtualization and service technologies to enable grid-computing," in *11th International Euro-Par Conference*, August 2005.

[7] A. M. Matsunaga, M. O. Tsugawa, S. Adabala, R. J. Figueiredo, H. Lam, and J. A. B. Fortes, "Science gateways made easy: the In-VIGO approach," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 6, pp. 905–919, April 2007.

[8] M. A. Murphy, M. Fenn, and S. Goasguen, "Virtual organization cluster model," in *17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, February 2009.

[9] M. Fenn, "A performance analysis of virtual machine monitors for use in the Open Science Grid," Honor's Thesis, December 2008, Calhoun Honors College, Clemson University.

[10] Open Science Grid. Http://www.opensciencegrid.org/.

[11] M. Ernst, P. Fuhrmann, M. Gasthuber, T. Mkrtchyan, and C. Waldmann, "dcache, a distributed storage data caching system," in *Computing in High Energy and Nuclear Physics*, 2001.

[12] Cyberinfrastructure Research Group. Http://cirg.cs.clemson.edu/.

[13] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency - Practice and Experience*, vol. 17, pp. 323–356, 2005.

[14] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "IP over P2P: Enabling self-configuring virtual IP networks for grid computing," in *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[15] Open Nebula. Http://www.opennebula.org/.

[16] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang, "Virtual clusters for grid communities," in *CCGrid 2006*, Singapore, May 2006.

[17] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual workspaces in the Grid," in *11th International Euro-Par Conference*, Lisbon, Portugal, September 2005.

[18] W. Emeneker and D. Stanzione, "Dynamic virtual clustering," in *IEEE Cluster 2007*, Austin, TX, September 2007.

[19] H. Nishimura, N. Maruyama, and S. Matsuoka, "Virtual clusters on the fly - fast, scalable, and flexible installation," in *CCGRID 2007: Seventh IEEE International Symposium on Cluster Computing and the Grid*, May 2007.

[20] J. A. B. Fortes, R. J. Figueiredo, and M. S. Lundstrom, "Virtual computing infrastructures for nanoelectronics simulation," *Proceedings of the IEEE*, vol. 93, no. 10, pp. 1839–1847, October 2005.

[21] J. Mugler, T. Naughton, and S. L. Scott, "OSCAR meta-package system," in *19th International Symposium on High Performance Computing Systems and Applications*, May 2005.

[22] M. J. Katz, P. M. Papadopoulos, and G. Bruno, "Leveraging standard core technologies to programmatically build Linux cluster appliances," in *Cluster 2002: IEEE International Conference on Cluster Computing*, April 2002.

[23] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "NPACI Rocks: Tools and techniques for easily deploying manageable Linux clusters," in *Cluster 2001: IEEE International Conference on Cluster Computing*, October 2001.

[24] P. M. Papadopoulos, C. A. Papadopoulos, M. J. Katz, W. J. Link, and G. Bruno, "Configuring large high-performance clusters at lightspeed: A case study," in *Clusters and Computational Grids for Scientific Computing 2002*, December 2002.

[25] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic virtual clusters in a grid site manager," in *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, June 2003.

[26] G. Bruno, M. J. Katz, F. D. Sacerdoti, and P. M. Papadopoulos, "Rolls: Modifying a standard system installer to support user-customizable cluster frontend appliances," in *IEEE International Conference on Cluster Computing*, September 2004.

[27] R. Davoli, "VDE: Virtual Distributed Ethernet," in *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom 2005)*, Trento, Italy, February 2005.

[28] A. I. Sundararaj and P. A. Dinda, "Towards virtual networks for virtual machine grid computing," in *Proceedings of the Third Virtual Machine Research and Technology Symposium*, San Jose, CA, May 2004.

[29] D. Wolinsky, A. Agrawal, P. O. Boykin, J. Davis, A. Ganguly, V. Paramygin, P. Sheng, and R. Figueiredo, "On the design of virtual machine sandboxes for distributed computing in Wide-area Overlays of virtual Workstations," in *First International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[30] P. Ruth, X. Jiang, D. Xu, and S. Goasguen, "Virtual distributed environments in a shared infrastructure," *Computer*, vol. 38, no. 5, pp. 63–69, 2005.

[31] J. Liu, Y. Li, N. V. Vorst, S. Mann, and K. Hellman, "A real-time network simulation infrastructure based on OpenVPN," *J. Syst. Softw.*, vol. 82, no. 3, pp. 473–485, 2009.

[32] P. O. Boykin, J. S. A. Bridgewater, J. S. Kong, K. M. Lozev, B. A. Rezaei, and V. P. Roychowdhury. (2007, September) A symphony conducted by Brunet. Online. [Online]. Available: http://arxiv.org/abs/0709.4048

[33] Google App Engine. Http://code.google.com/appengine/.

[34] P. Saint-Andre and R. Meijer, "Streaming XML with Jabber/XMPP," *IEEE Internet Computing*, vol. 9, no. 5, pp. 82–89, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.6260&rep=rep1&type=pdf

[35] L. Stout, M. Murphy, and S. Goasguen, "Kestrel: An xmpp-based framework for many task computing applications," in *2nd Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS 2009)*, Portland, OR, November 2009.

[36] J. Wagener, O. Spjuth, E. Willighagen, and J. Wikberg, "XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous Web services," *BMC bioinformatics*, vol. 10, no. 1, p. 279, 2009. [Online]. Available: http://www.biomedcentral.com/content/pdf/1471-2105-10-279.pdf

[37] J. H. Christensen, "Using RESTful web-services and cloud computing to create next generation mobile applications," in *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. New York, NY, USA: ACM, 2009, pp. 627–634.

[38] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the Intercloud-Protocols and Formats for Cloud Computing Interoperability," in *Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services-Volume 00*. IEEE Computer Society, 2009, pp. 328–336.

[39] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A Berkeley view of cloud computing," University of California at Berkeley, Technical Report EECS-2009-28, February 2009. [Online]. Available: http://nma.berkeley.edu/ark:/28722/bk000471b6t

[40] R. Buyya, C. Yeo, and S. Venugopal, "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*, 2008. [Online]. Available: http://arxiv.org/pdf/0808.3558

[41] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual clusters," in *eScience 2008*, Indianapolis, IN, December 2008.

[42] M. Fenn, M. A. Murphy, and S. Goasguen, "A study of a KVM-based cluster for grid computing," in *47th ACM Southeast Conference (ACMSE '09)*, Clemson, SC, March 2009.

[43] I. Habib, "Virtualization with KVM," *Linux Journal*, vol. 2008, no. 166, p. 8, February 2008.

[44] L. van Doorn, "Hardware virtualization trends," in *Second International Conference on Virtual Execution Environments*, June 2006.

[45] K. Keahey, K. Doering, and I. Foster, "From sandbox to playground: Dynamic virtual environments in the Grid," in *5th International Workshop on Grid Computing (Grid 2004)*, November 2004.

[46] 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 *Symposium on Operating Systems Principles (SOSP '03)*, 2003.

[47] A. Whitaker, M. Shaw, and S. D. Gribble, "Denali: Lightweight virtual machines for distributed and networked applications," University of Washington, Tech. Rep. 02-02-01, 2002.

[48] M. Fenn, J. Lauret, and S. Goasguen, "Contextualization in practice: The Clemson experience," in *13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2010)*, Jaipur, India, February 2010.

[49] M. Fenn. (2009, December) Converting a partition image to a disk image. Http://www.mfenn.com/converting_a_partition_image_to_a_disk_image.

[50] Stoker. Http://cirg.cs.clemson.edu/software/stoker/.

[51] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, "Dynamic provisioning of Virtual Organization Clusters," in *9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '09)*, Shanghai, China, May 2009.

[52] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in P2P systems," *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, February 2003.

[53] Adium. Http://adium.im/.

[54] xmpppy. Http://xmpppy.sourceforge.net/.

[55] M. Fenn, M. A. Murphy, and S. Goasguen, "An evaluation of KVM for use in cloud computing," in *2nd International Conference on the Virtual Computing Initiative (ICVCI '08)*, May 2008.

[56] R. Bisseling and L. Loyens, "Towards peak parallel linpack performance on 400," *Supercomputer*, vol. 45, pp. 20–27, 1991.

[57] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC Challenge (HPCC) benchmark suite," in *Supercomputing '06*, 2006.

[58] M. Matsuda, T. Kudoh, and Y. Ishikawa, "Evaluation of MPI implementations on grid-connected clusters using an emulated WAN environment," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid03)*, 2003.

[59] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. C. Fox, and A. J. G. Hey, Eds. Wiley, 2003, ch. 11, pp. 299–350.

[60] (2009, October) ACM policy and procedures on plagiarism. Association for Computing Machinery. Http://www.acm.org/publications/policies/plagiarism_policy.