Distributed Management of Virtual Cluster Infrastructures

Michael A. Murphy, Michael Fenn, Linton Abraham,
Joshua A. Canter, Benjamin T. Sterrett, and Sebastien Goasguen
School of Computing
Clemson University
Clemson, South Carolina 29634-0974 USA
{mamurph, mfenn, labraha, jcanter, bsterre, sebgoa}@clemson.edu

Abstract

Cloud services that provide virtualized computational clusters present a dichotomy of systems management challenges, as the virtual clusters may be owned and administered by one entity, while the underlying physical fabric may belong to a different entity. On the physical fabric, scalable tools that "push" configuration changes and software updates to the compute nodes are effective, since the physical system administrators have complete system access. However, virtual clusters executing atop federated Grid sites may not be directly reachable for management purposes, as network or policy limitations may prevent unsolicited connections to virtual compute nodes. For these systems, a distributed middleware solution could permit the compute nodes to "pull" updates from a centralized server, thereby permitting the management of virtual compute nodes that are inaccessible to the system administrator. This paper compares both models of system administration and describes emerging software utilities for managing both the physical fabric and the virtual clusters.

1. Introduction

Through the use of virtualization technology, cloud computing systems provide a mechanism by which large-scale computational resources can be provided to domain scientists, freeing individual scientific users from the need to purchase and maintain physical systems and infrastructure. Domain scientists can access and use these cloud systems via grid computing interfaces, thereby allowing individual physical sites to support different user groups. An immediate challenge results from this resource sharing, due to differences in software requirements between scientific applications. Cloud sites must either provide a mechanism to install a wide range of libraries and applications for different user groups, or users must adapt their applications to the environments made available on the computational sites.

One possible solution to this software provisioning problem is to utilize Virtual Organization Clusters [1] to separate the virtualized cloud systems presented to end users from the underlying physical hardware. Grid systems designed around the Virtual Organization Cluster (VOC) Model provide a separate administrative domain for each Virtual Organization (VO), or group of affiliated users [2], thereby allowing each VO to customize its software environment. VOC administrators can install software and set configuration policies that are appropriate for the domain scientists they support, without site-specific limitations imposed by the physical fabric. Each VO is thus able to provide the software packages desired by its member scientists, while physical site administrators are freed from maintaining different software sets for each group of end users. Figure 1 presents such a use case.

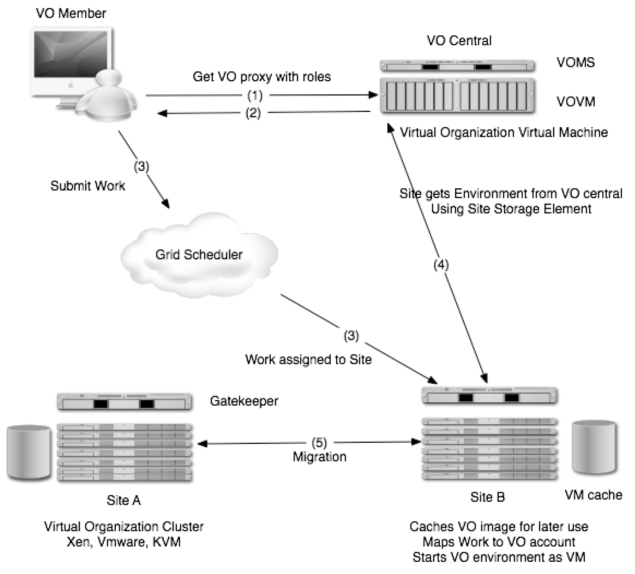

Figure 1. Virtual Organization member utilizing a Virtual Organization Cluster

In order to make systems designed around the VOC Model practical for both end users and physical fabric providers, mechanisms must be available to configure and manage both the cloud systems and the underlying physical fabric. Although VOCs may be dynamically expanded and shrunk in size, their core Virtual Machine (VM) images are designed to be used for long time periods on the order of weeks, months, or perhaps even years. These VOC compute node images are neither ephemeral nor disposable; rather, they are virtualized equivalents of physical hardware dedicated to a specific scientific mission. Thus, management mechanisms for VOCs must exist to reconfigure, update, and add software to the images. Since VOCs are hosted on third-party physical systems with unique site policies and network architectures, there is no guarantee that the VO will be provided with direct access to the VOC compute node image to make post-deployment modifications, rendering traditional direct administration techniques ineffective. Physical fabric sites, in contrast, can be administered directly, since the physical administrators have direct access to the hardware. Scalable and flexible approaches must still be used, however, since individual physical nodes will be in use at different times as the sizes of individual VOCs change to respond to computational load.

This paper presents two divergent mechanisms for system administration, which are complementary when used to manage cloud computing systems designed according to the VOC Model. For management of the physical fabric, a scalable application called Stoker [3] pushes configuration changes to physical nodes via standard network transports. VOCs are managed by the second system, an application in the early design stage called Pulley, which periodically requests configuration updates from a central server and is designed to operate in situations where there is no direct access to the systems being administered. Together, these opposing mechanisms for systems management can be utilized to configure, update, and maintain computational clouds over long temporal periods.

The remainder of this paper is organized as follows: Section 2 presents related work, after which section 3 provides a brief overview of the Virtual Organization Cluster Model. Section 4 presents Stoker and discusses management of the physical hardware, while section 5 is devoted to the management of the VMs and the initial design of Pulley. Test results of running Stoker are presented in section 6, after which section 7 concludes and discusses future work.

2. Related Work

Virtualization of entire clusters was first proposed in [4], and it has been subsequently realized using different models, such as Globus Virtual Workspaces [5], [6], In-VIGO [7], Dynamic Virtual Clustering [8], VMPlants [9], and Virtual Clusters on the Fly [10]. Rapid re-provisioning of physical systems, as an alternative to using virtualization technologies, has been studied in the Cluster-On-Demand (COD) system [11]. Each of these models has focused on rapid provisioning of virtual clusters with short lifespans, typically on the order of a single job. As a result, reconfiguration and on-going maintenance of long-lived virtual clusters has not been studied, even though tools and techniques have long been available for managing regular computational clusters that run jobs directly on the hardware.

Post-installation configuration management is a feature of rapid physical cluster construction tools such as ROCKS [12] and OSCAR [13]. Software and configuration changes can be made through several mechanisms, such as the addition of ROCKS rolls [14], or the use of remote command execution systems such as the Cluster Command and Control Suite [15] or Tentakel [16] to push changes to the compute nodes. Alternatively, nodes can be completely re-imaged using a tool such as the System Installation Suite [17]. All these "traditional" techniques are best suited for the management of physical systems within a single site. Moreover, with the exception of the ability of the Cluster Command and Control Suite to handle multiple individual clusters, these tools tend to focus on batch management of entire systems simultaneously, omitting fine-grained targeting of individual systems.

Another mechanism for post-installation management of systems is Cfengine [18], which utilizes a descriptive language to configure autonomous agents. These agents communicate with a central policy server to effect configuration changes via a convergent process [19] whereby each machine tends to move toward a desired state. Although Cfengine allows different groups of systems to be targeted by specifying policies around system classes, it is still designed around the concept of single-site management. Cfengine also requires dedicated services and transports to operate, which might cause difficulties when spanning grid sites.

The primary contribution of this paper is to identify the challenges associated with maintaining long-lived virtual clusters that are not tied to specific hardware systems, as well as to document progress made to date on utilities to manage these distributed systems. This work differs from previous research by investigating scalable long-term management techniques with a dynamic model of cluster virtualization that does not require complete re-construction of the cluster on a per-job basis.

3. Virtual Organization Cluster Model

The Virtual Organization Cluster Model [1], illustrated in figure 2, specifies the design of both physical and virtual systems that enable Virtual Organizations to run dedicated clusters of virtual machines, termed Virtual Organization Clusters (VOCs). Each VOC represents an isolated administrative domain, within which a VO has extensive control over the choice of operating system, libraries, and scientific software. Within the bounds of minor operational constraints, described below, the VO also is able to configure the virtual cluster as desired and set low-level operational and usage policies.

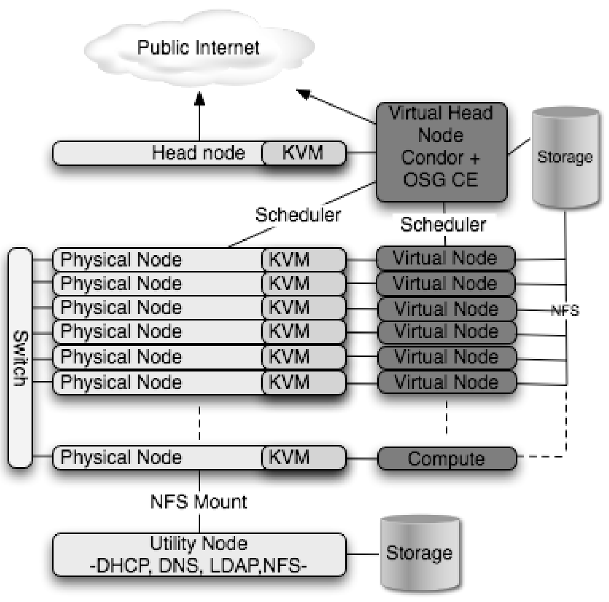

Figure 2. A Virtual Organization Cluster hosted on a physical cluster utilizing the Kernel-based Virtual Machine virtual machine monitor

VOCs are executed atop physical computing fabric made available by affiliated organizations, and perhaps third parties, over a standard grid computing platform such as the Open Science Grid [20]. Each of these physical sites is also an isolated Physical Administrative Domain (PAD), managed independently from the VOCs it hosts. Figure 3 illustrates a single physical site hosting multiple VOCs. The physical hosts, virtualization systems, network infrastructure, and shared Grid services comprise the PAD in this example.

The VOC Model provides a means by which each VO can have its own dedicated cluster and associated Virtual Administrative Domain (VAD), as illustrated in figure 3. Virtual Organization Clusters using this model are formed from VMs that are either created at the physical site where they are to be used, created by grid middleware (for example, In-VIGO [7]), or manually transmitted to the physical site by a VO administrator. Unlike systems that utilize one virtual disk image per VM instance [5], [8], [10], VOCs are explicitly designed to work with either one or two disk image files: a single image representing the configuration of a compute node, and an optional second image that contains a cluster head node. All compute node VMs are spawned from the single compute node image using a copy-on-write mechanism that allows the original image file to be read-only. The result of this design is that both disk space and network bandwidth requirements are reduced, while management of the VOC is simplified.

A major benefit of the VOC Model design is that few constraints are placed on the VM. The VO has great flexibility in selecting the operating system and software environment best suited to the requirements of its users. Of the few constraints that do exist, the primary ones are as follows:

- **Image Compatibility.** The VM image must be in a format usable by the Virtual Machine Monitor (VMM) or hypervisor software in use at the physical site(s) where the VOC will be executed.
- **Architecture Compatibility.** The operating system running in the VM must be compatible with the system architecture exposed by the VMM or hypervisor.
- **Dynamic Reconfigurability.** The guest system inside the VM must be able to have certain properties, such as its MAC address, IP address, and hostname, set at boot time.
- **Scheduler Compatibility.** When only a single image file is used with a shared scheduler provided by the physical site, the scheduler interface on the VM must be compatible with the shared scheduler.

For the purpose of discussing dynamic provisioning and scheduling of jobs on VOCs, the single-image case is simpler, since there is only one job queue to be maintained at any location on the system. The implementation described in this paper thus assumes that each VO provides only a compute node image, which has the required scheduler client installed and operable.

4. Management of Physical Machines

In order to instantiate Virtual Organization Clusters, underlying physical hardware is needed to run virtualization software and host the VOC nodes. While this hardware could be owned by the Virtual Organizations themselves, a cloud computing model would favor the use of specialized hosting providers that make physical fabric resources available on a grid system. Each hosting provider would thus be responsible for the acquisition and maintenance of all the hardware necessary to construct physical clusters, including the computing systems, local storage, network systems, power conditioning and distribution, and cooling infrastructure. Physical site administrators would be responsible for local site policies and maintenance tasks, but their administrative responsibility would extend only to the physical fabric at the specific site. Thus, each individual physical site would be a separate administrative domain, known as a Physical Administrative Domain (or PAD).

A simplifying assumption made about each physical site is that physical resources are devoted exclusively to hosting VOCs: no scientific jobs are executed directly on the hardware. As a result, the software set required on each compute node is minimal and consists of the host operating system, virtualization applications, and any needed driver software to support installed hardware. Standard networking technologies, such as the Dynamic Host Configuration Protocol

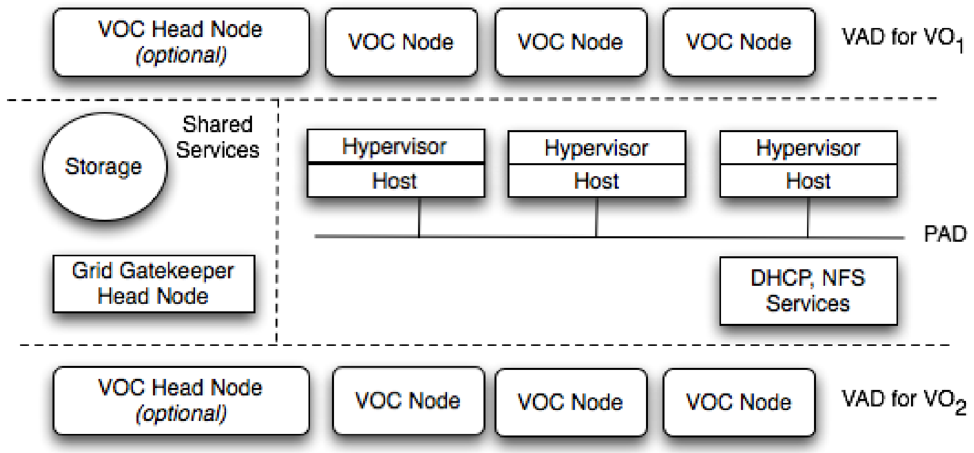

Figure 3. Two Virtual Organization Clusters on a single physical site

(DHCP) and Domain Name System (DNS), are employed in a centralized manner to enable connectivity between systems, while simultaneously maintaining the scalability of the physical cluster as a whole. By centralizing host-specific settings, such as the IP address and host name of each compute node, management inside the PAD is simplified.

One issue that does arise when using standard networking services is the replication of identical information across different services. For example, if fixed IP and host name assignments are to be made using DHCP, then the same mapping of host names to IP addresses must be entered into the DNS records for resolution of host information to work correctly within the system. A solution to avoiding this replication is to centralize the duplicate information, using an external database such as a Lightweight Directory Access Protocol (LDAP) server. Services utilizing the information in the database are then configured to use the external database directly, provided the services have the required integration capability. Otherwise, the services are adapted, through the use of middleware layers, to update their local configurations from the centralized database upon request.

4.1. Stoker

Stoker is a scalable remote management tool, whose overall architecture is shown in figure 4. Stoker differs from prior management tools such as Tentakel [16] and the Cluster Command and Control Suite [15] in that it can obtain system information directly from a centralized database with system grouping capabilities, thereby avoiding replication of host information in a configuration file. Stoker also has an extensible, modular design with three major components: the *warehouse(s), core*, and *actor(s)*. These components handle retrieving contact information for a node or group of nodes, spawning and joining actor threads, and performing some type of action on the target nodes. Each component will be covered in more detail in sections 4.1.1, 4.1.2, and 4.1.3.

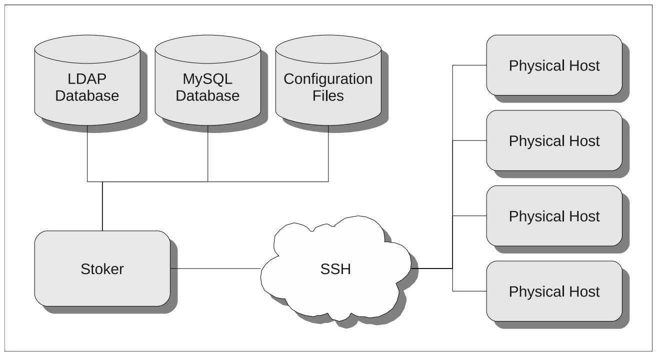

Figure 4. Overview of Stoker

4.1.1. Stoker Warehouses. Stoker can use multiple data sources or *warehouses* to gather data about target nodes. Central to the warehouse data retrieval task is the concept of a *resolver*. Resolvers take as input a logical node or group name and return a data structure containing addressing information potentially including, but not limited to, hostname(s), IP address(es), and MAC address(es). Stoker's grouping feature allows machines to be organized into arbitrary groups and subgroups for the convenience of the administrators. Sample groups include: "all odd-numbered machines", "all machines with the Apache web server installed", etc.

A separate resolver is required for each type of data warehouse. Resolvers currently exist for LDAP databases, MySQL databases, and regular configuration files. An administrator wishing to implement a new resolver needs only to write a small Python class that retrieves the information from the warehouse and inserts it into a simple data structure defined by Stoker.

4.1.2. Stoker Core.

A design goal of the Stoker *core* was to encapsulate the complexity inherent in a multi-threaded application, thus allowing a system administrator to extend or create, with minimal effort, new warehouses and actors to meet his needs. The primary function of the core is to invoke the resolver(s) and spawn an actor thread for each target. The core then manages each thread, collecting any output from the actor, and generates an activity report for the user.

Since the core does not have *a priori* knowledge of whether or not a user-supplied target is a single node or a group of nodes, it must be flexible enough to handle a situation in which the user inadvertently specifies a target multiple times. For example, if a user specifies, "toggle SELinux enforcing state on webserver01, all_apache_servers," webserver01 may very well be a member of the all_apache_servers group; however, the user almost certainly did *not* intend to toggle the state of webserver01 *twice*.

4.1.3. Stoker Actors.

Stoker *actors* are analogous to Stoker warehouses in the sense that they are (potentially) simple scripts that perform a simple task at the direction of the core. Actors execute in their own threads and have well-defined data structures that specify all known information about a single target. The actor's task is to apply its argument, also provided by the core, to its target. Potential actors include such tasks as remote command invocation via SSH, ping, Wake-on-LAN, and local (relative to the user's machine) command invocation. New actors are easily created by the system administrator to perform new functions.

4.2. DHCP and DNS Middleware

When managing physical clusters, it is convenient to have a single repository for all node-specific information. Unfortunately, many software packages do not include support for a centralized configuration repository, requiring instead a product-specific configuration file. To eliminate needless data duplication and possible inconsistencies between services that share data, database-aware middleware was developed (illustrated in figure 5) using a Lightweight Directory Access Protocol (LDAP) server as the data source. As a proof of concept, two network services were adapted to use this middleware: ISC *dhcpd*, a Dynamic Host Configuration Protocol (DHCP) server; and *dnsmasq*, a Domain Name System (DNS) server.

The middleware is implemented as a Python script that accesses the LDAP database and writes a configuration file for the given software package. This script is then integrated with the service initialization scripts so that every time the software package is started or restarted, the configuration file will be regenerated from the LDAP database, ensuring that any changes will be propagated throughout the system.

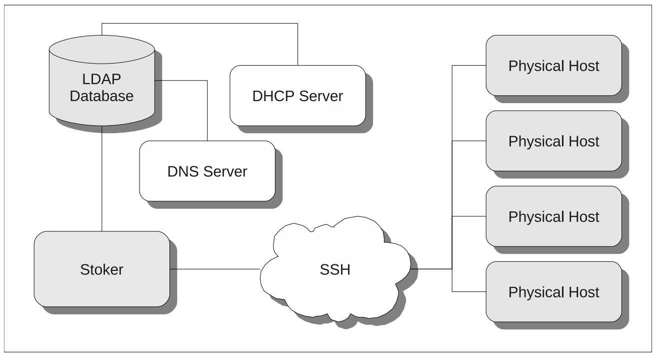

Figure 5. Automatic integration of Stoker with DHCP and DNS services

5. Management of Virtual Machines

Unlike the physical fabric systems, which are comprised of single, isolated sites, the Virtual Organization Clusters potentially execute on arbitrary sites, or even across physical sites. As a result, it is not practical to assume that virtual machines will necessarily always be directly reachable, since physical sites may be constructed using private networks with Network Address Translation (NAT) used on routers connecting the private networks to the commodity Internet. Without complex arrangements for forwarding connections to individual VOC nodes, the use of client-initiated transports like SSH is not feasible. As a result, configuration tools that operate on a "push" model, such as Stoker, are not suited for managing these virtual clusters.

An alternate system that can traverse NAT boundaries automatically is in the initial design stages. This middleware utility, called Pulley, will use a central database and application server to publish virtual cluster node configurations on the Internet. Client VMs will periodically poll the application server using standard HTTP Web service mechanisms. Since the polling requests will originate from the VMs themselves, they will be able to traverse NAT and firewall boundaries at the physical sites, without requiring any additional software or services to be made available on the physical fabric. This design is illustrated in figure 6.

Since Pulley is in a formative stage of design, the exact mechanism of its management functionality is not yet known. Two models of operation are planned for investigation. One of these possible architectures would be to adapt Stoker actors to draw operational information directly from the configuration database via middleware, instead of relying on human system administrators to provide command input. The second potential architecture would be to implement a policy-based mechanism such as Cfengine, adapting the agent-based system to operate across Grid sites via middleware protocols. It is even possible that a hybrid of both architectures will be used in the final system.

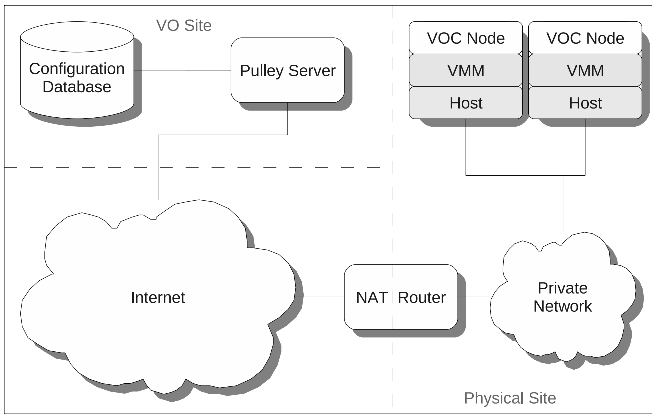

Figure 6. Overview of Pulley

6. Stoker Test Results

In order to measure overhead created by the resolution and threading management processes and to assess the performance improvement of parallelizing the command execution process, several tests were conducted using Stoker. The first test, with results shown in figure 7, parallelized a short-running `/bin/hostname` task. Parallel execution was found to be of limited utility beyond 4–6 threads in this case due to the extremely short runtime of the program ($\approx 0.001s$) relative to the total time necessary to spawn a thread and execute a remote command via SSH ($\approx 0.225s$). This test was conducted on a low-latency network inside a private computing cluster.

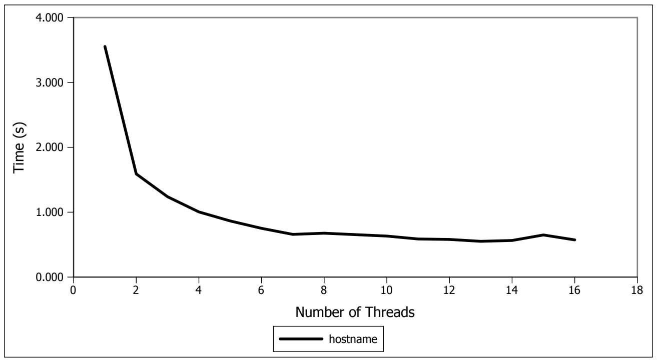

Figure 7. Stoker performance on a low-latency 16-node cluster when parallelizing a hostname job

The next test (figure 8) involved a much longer running job: a ten-second process sleep that was engineered to simulate the restart procedure of a network service. This test was conducted on the same private, low-latency cluster as the first test. Performance improvements were observed up to the limit of one thread per target (16 nodes in this case). It should be noted that no performance improvement was measured for 8–15 threads.

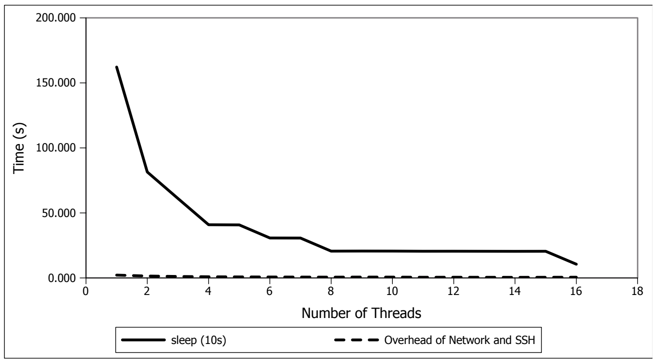

Figure 8. Stoker performance on a low-latency 16-node cluster when parallelizing a 10-second sleep operation

Since the 10-second sleep procedure had a known runtime, network and SSH overhead could be measured. As shown in figure 9, these overheads generally decreased in an absolute sense until 14 threads were utilized. Beyond this point, the overhead of spawning new threads began to increase. However, overhead as a percentage of the total time remained constant until 16 threads, the limit for this test, were utilized.

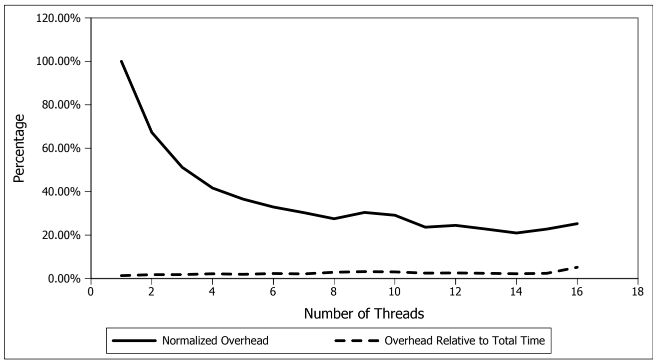

Figure 9. Comparative overheads of network and SSH with respect to the remote job being executed on a low-latency 16-node cluster. The normalized overhead is expressed as a percentage of the maximum observed overhead, which occurred when the number of threads was equal to 1. Relative overhead is expressed as the percentage of network and SSH overhead present in the total execution time of a 10-second sleep job.

To determine whether overheads would be more significant across a higher latency network, another 10-second sleep test was conducted on 27 public laboratory workstations. Results of this test, summarized in figure 10, showed improvements in performance as threads increased from 1 to 6. Performance improvements declined beyond 6 threads, even through 27 machines were targeted with the 10-second sleep job. After further testing, including manual execution of the same job using shell scripts instead of Stoker, it was determined that SSH authentication latency was increasing as the number of simultaneous SSH processes was increased. The root cause of this behavior was found to be serialization of the SSH authentication requests resulting from the use of a single Network File System (NFS) share for storing the SSH keys used for authentication. Since the single NFS server was also utilized for unrelated purposes, additional latency was added to the authentication process. As shown in figure 11, overhead as a percentage of total execution time increased with increasing parallelism, effectively negating the benefits of additional Stoker actor threads.

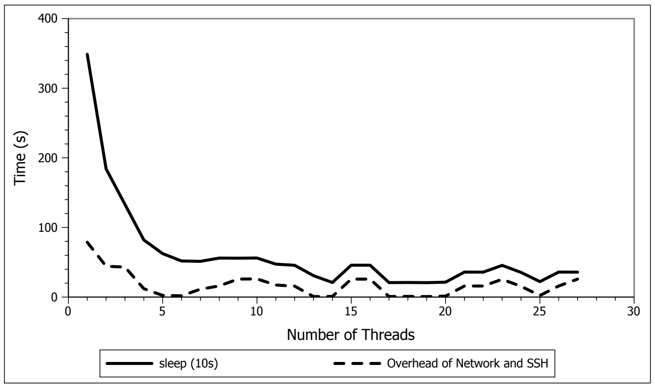

Figure 10. Stoker performance on a high-latency group of 27 public laboratory workstations

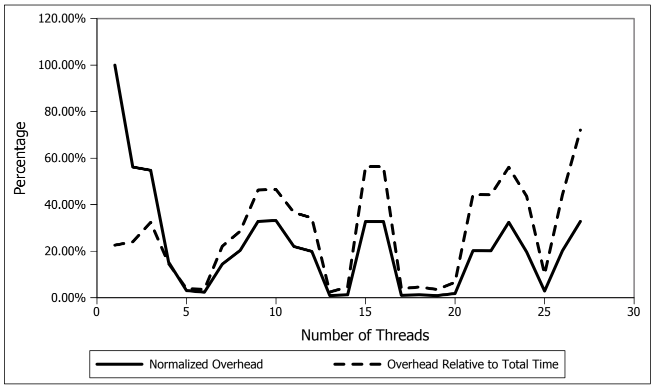

Figure 11. Comparative overheads of network and SSH with respect to the remote job being executed on a high-latency 27-node group of public workstations.

7. Conclusions and Future Work

Virtual Organization Clusters have the potential to enable the execution of long-lived clusters of virtual machines, which may span Grid sites to make best use of available physical fabric to provide cloud computing resources for e-Science applications. Management of the individual physical grid sites may be accomplished using scalable push-oriented tools like Stoker, but management of the virtual clusters will require a new mechanism that avoids imposing additional requirements on the physical systems. A prototype of such a system, called Pulley, will be the subject of future research into distributed management of systems that are not directly accessible.

Acknowledgments

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. The authors would like to thank Heather Harton, Hanjun Xiang, Hardic Barot, Justin Garrick, Kyle Peacock, Vikas Patel, and Nisarg Gandhi for their assistance in testing Stoker and adding ROCKS MySQL database support. Also, the authors would like to thank Scott Duckworth for his assistance in testing SSH authentication on the public laboratory machines.

References

[1] M. A. Murphy, M. Fenn, and S. Goasguen, "Virtual organization clusters," in *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009)*, Weimar, Germany, February 2009.

[2] 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.

[3] "Stoker home page." [Online]. Available: http://cirg.cs.clemson.edu/software/stoker/

[4] 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.

[5] 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.

[6] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *Proceedings of the 17th international symposium on High Performance Distributed Computing (HPDC 2008)*, 2008.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] R. Flanery, A. Geist, B. Luethke, and S. L. Scott, "Cluster command & control (c3) tools suite," in *3rd Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS 2000)*, 2000.

[16] "Tentakel homepage," 2007. [Online]. Available: http://tentakel.biskalar.de/

[17] S. Dague, "System installation suite: Massive installation for linux," in *Ottawa Linux Symposium*, 2002.

[18] M. Burgess and R. Ralston, "Distributed resource administration using cfengine," *Software – Practice and Experience*, vol. 27, no. 9, pp. 1083–1101, September 1997.

[19] M. Burgess, "A tiny overview of Cfengine: Convergent maintenance agent," in *First International Workshop on Multi-Agent and Robotic Systems*, 2005.

[20] "Open Science Grid." [Online]. Available: http://www.opensciencegrid.org