Cluster Computing manuscript No.
(will be inserted by the editor)

Experiences with Self-Organizing, Decentralized Grids Using the Grid Appliance

David Isaac Wolinsky · Panoat Chuchaisri · Kyungyong Lee · Renato Figueiredo

Received: date / Accepted: date

Abstract "Give a man a fish, feed him for a day. Teach a man to fish, feed him for a lifetime" – Lau Tzu

Large-scale grid computing projects such as TeraGrid and Open Science Grid provide researchers vast amounts of compute resources but with requirements that could limit access, results delayed due to potentially long job queues, and environments and policies that might affect a user's work flow. In many scenarios and in particular with the advent of Infrastructure-as-a-Service (IaaS) cloud computing, individual users and communities can benefit from less restrictive, dynamic systems that include a combination of local resources and on-demand resources provisioned by one or more IaaS provider. These types of scenarios benefit from flexibility in deploying resources, remote access, and environment configuration.

In this paper, we address how small groups can dynamically create, join, and manage grid infrastructures with low administrative overhead. Our work distinguishes itself from other projects with similar objects by enabling a combination of decentralized system organization and user access for job submission in addition to a web 2.0 interfaces for managing grid membership and automate certificate management. These components contribute to the design of the "Grid Appliance," an implementation of a wide area overlay network of virtual workstations (WOW), which has developed over the past six years into a mature system with several deployments and many users. In addition to an architectural description, this paper contains lessons learned during the development and deployment of "Grid Appliance" systems and a case study backed by quantitative analysis that verifies the utility of our approach.

Keywords Grid, Cloud, P2P Grid

D. Wolinsky
Yale Univeristy
E-mail: david.wolinsky@yale.edu

P. Chuchaisri
University of Florida
E-mail: pchuchai@cise.ufl.edu

K. Lee
University of Florida
E-mail: klee@acis.ufl.edu

R. Figueiredo
University of Florida
E-mail: renato@acis.ufl.edu

1 Introduction

Grid computing presents opportunities to combine distributed resources to form powerful systems. Due to the challenges in coordinating resource configuration and deployment, researchers tend to either become members of existing grids or deploy their own private resources. The former approach is limited by lack of flexibility in the environment and policies, while the latter requires expertise in systems configuration and management. Though there exists a wealth of middleware available, including resource managers such as Condor [22], Torque (PBS) [26], and Sun Grid Engine [32], many see the cost of installing and managing these systems as being greater than their usefulness and as a result turn to inefficient ad hoc resource discovery and allocation. To combine resources across multiple domains solutions there exist solutions such as the Globus Toolkit [12] or gLite [6]; however, these tool sets come with their own challenges that require the level of expertise most researchers in fields outside of information technology lack.

With the recent advent of cost-effective on-demand computing through Infrastructure-as-a-Service "clouds",

new opportunities for user-deployed grids have arisen; where, for example, a small local computer cluster can be complemented by dynamically provisioned resources that run "cloud-burst" workloads. However, while cloud-provisioned resources solve the problem of on-demand instantiation, configuring resources to seamlessly and securely integrate with one's infrastructure remains a challenge. In particular, users may provision resources from multiple IaaS providers as well as their own local resources in order to avoid relying on one particular vendor, a painful lesson learned by those using Amazon's EC2 exclusively on April 21, 2011 [1]. On that day and for some days ensuing, Amazon's web services were unavailable rendering many web sites out of commission. A well balanced approach to avoid these types of dependencies results in a system with similar configuration demands to that of a distributed grid: while a cloud image can be encapsulated with a grid computing stack, it still needs configuration in terms of allocating and distributing the appropriate certificates, network configuration to establish end-to-end connectivity, and proper configuration of the middleware to establish worker, submit, and scheduler nodes.

In this paper, we present techniques that reduce the entry barrier in terms of necessary expertise and time investment in deploying and extending ad hoc, distributed grids. To verify this assertion, we have implemented a system supporting these ideas in the "Grid Appliance," which as will be demonstrated, allows users to focus on making use of a grid while minimizing their efforts in setting up and managing the underlying components. The core challenges solved by our approach include:

- decentralized directory service for organizing grids,
- decentralized job submission,
- grid single sign on through web services and interfaces,
- sandboxing with network support,
- and all-to-all connectivity despite network asymmetries.

As an extension to our previous work [37], this paper presents updates to the preceding topics, additional lessons learned, and a discussion on the use of the "Grid Appliance" as a utility for ondemand construction of parallel job environments.

The "Grid Appliance" project and concepts have been actively developed and used in several projects for the past six years. Of these projects, Archer, a distributed grid for computer architecture research, has demonstrated the feasibility and utility of this approach

[1] `http://money.cnn.com/2011/04/21/technology/amazon_server_outage/index.htm`

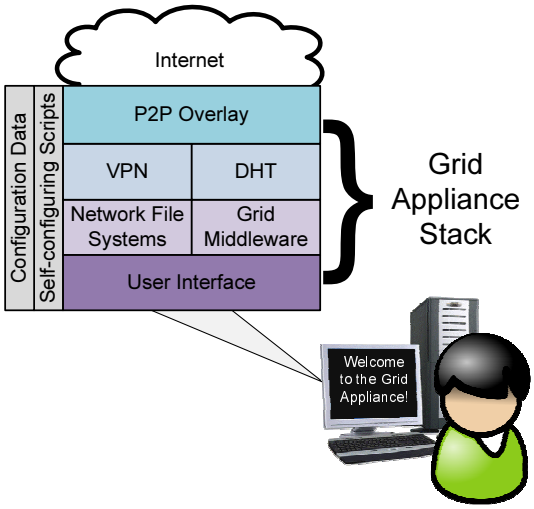

Fig. 1 The "Grid Appliance" connects to other resources over a common network. Both the grid middleware and the VPN use the P2P overlay to configure and connect the user to other members of the grid. The process uses configuration data provided by the web interface to self-configure the system using information available in the P2P network.

by deploying a shared collaborative infrastructure spanning clusters across six US universities, where the majority of the nodes are constrained by network address translation (NAT). Every resource in Archer is configured in the same, simple manner: by deploying a "Grid Appliance" that self-configures to join a wide-area grid. Researchers interested or desiring the ability to access both grid resources and specialized commercial simulation tools (such as Simics) can easily use and contribute resources from this shared pool with little effort by joining a website, downloading a configuration image and a virtual machine (VM), and starting the VM inside a VM manager (VMM). Upon completion of the booting process, users are connected to the grid and able to submit and receive jobs.

The "Grid Appliance" has more recently been used as an academic tool for exploring parallel computing through the Future Grid project. MPI (Messaging Passing Interface) and Hadoop are excellent utilities for solving problems using parallel computation and as potential research avenues; however, configuration and resource requirements present challenges that may limit experiences with these tools. To address this, we have designed a self-configuring environment for deploying parallel environments for use within the "Grid Appliance." By incorporating the system into Future Grid, users can easily deploy their MPI and Hadoop clusters allowing them to focus on user interaction with these systems as opposed to management issues.

At the heart of our approach lies a P2P infrastructure based upon a distributed hash table (DHT) useful for decentralized configuration and organization of sys-

tems. Peers are able to store key, value pairs into the DHT and to query the DHT with a key and potentially receive multiple values efficiently. The DHT provides discovery and coordination primitives for the configuration of a decentralized P2P virtual private network (VPN), which supports unmodified applications across a network overlay. The DHT is also used for the decentralized coordination of the grid. Users can configure their grid through a web interface, which outputs configuration files that can be used with the "Grid Appliance."

The techniques described in this paper have many applications. The basic system supports the creation of local grids by starting a virtual machine on the computers intended for use within the grid and using LAN multicast for discovery. It allows users to seamlessly combine their dedicated grids with external resources such as workstations and cloud resources. The level of familiarity with security, operating systems, and networking is minimal as all the configuration details are handled as components of the system. Management of the system including users and network configuration utilizes a social networking like group interface, while deployment uses pre-built virtual machine images. A graphical overview of the system is illustrated in Figure 1.

These techniques simplify the tethering of resources across disparate networks. The setup of security, connectivity, and their continuous management imposes considerable administrative overhead, in particular when networks are constrained by firewalls and NAT devices that prevent direct communication with each other, and which are typically outside the control of a user or lab. Our approach integrates decentralized systems behind NATs in a manner that does not require the setup of exceptions and configuration at NAT/firewall by system administrators.

The rest of the paper is as follows. Section 2 highlights our previous work to provide background for our contributions in this paper. In Section 3, we describe the components of our "Grid Appliance" WOW. Section 4 provides a case study of a grid deployment using standard grid deployment techniques compared to our "Grid Appliance," describing qualitatively the benefits and evaluating quantitatively the overheads of this approach. We share our experiences from this long running project in Section 6. Finally, Section 7 compares and contrasts other solutions to these problems.

2 WOWs

This work furthers the vision began by our earlier concepts on wide-area overlays of virtual workstations [14] (WOW). The WOW paper established the use of virtualization technologies, primarily virtual networking and virtual machines, to support dynamic allocation of additional resources in grids that span wide area networks. For reference, the extensions made in this paper to the WOW concept are means for the dynamic creation of grids with support for security, decentralized access, and user-friendly approaches to grid management. This section covers the development of WOWs over the years as it relates to our other publications and as means to distinguish the contributions made in this paper.

2.1 P2P Overlays

Peer-to-peer or P2P systems create environments where members have a common functionality. P2P systems are often used for discovery in addition to some user-specific service, such as voice and video with Skype or data sharing with BitTorrent. Many forms of P2P have autonomic features such as self-healing and self-optimization with the ability to support decentralized environments. As we will show, this makes their application in our system very attractive.

For the "Grid Appliance," we make use of Brunet [7], a type of structured overlay. Structured overlays tend to be used to construct distributed hash tables (DHT) and in comparison to unstructured overlays provide faster guaranteed search times ($O(\log N)$ compared to $O(N)$, where N is the size of the network). The two most successful structured overlays are Kademlia [24], commonly used for decentralized BitTorrent, and Dynamo [8], to support Amazon's web site and services, while another two, Chord [31] and Pastry [28], dominate academic proceedings.

Brunet's support for NAT traversal and seamless bootstrapping [40] distinguishes it from other structured overlays. In WOWs [14], Brunet facilitated dynamic IP links amongst peers in the grid. Since then, it has been extended to support DHT with atomic operations for decentralized DHCP (Dynamic Host Configuration Protocol) [16], efficient relays when direct NAT traversal fails [38], resilient overlay structure and routing [15], and cryptographically secure messaging [38].

2.2 Virtual Private Networks

A common question with regards to this work is "why VPNs?" The core reason is connectivity. IPv4 has a limited address space, which has been extended through the use of NAT allowing a single IP to be multiplexed by multiple devices. This creates a problem; however, as

it breaks symmetry in the Internet limiting the ability for certain peers to become connected and which peers can initiate connections. With the advent of IPv6, the situation might improve, but there are no guarantees that NATs will disappear nor can users be certain that firewalls will not be in place that inhibit symmetry. A VPN circumvents these issues, so long as the user can connect to the VPN, as all traffic is routed through a successfully connected pathway.

The problem with traditional VPN approaches is management overhead including maintaining resources on public IP addresses and establishing links amongst members in the VPN. The VPN used in the system is called IPOP [38,13]. IPOP (IP over P2P), as the name implies, uses a P2P overlay (Brunet) to route IP messages. By using P2P, maintaining dedicated bootstrap nodes have less overhead, our approach with IPOP allows an existing Brunet infrastructure to bootstrap independent Brunet infrastructures in order to isolate IPOP networks in their own environments [40].

Once IPOP has entered its unique Brunet overlay, it obtains an IP address. IP address reservation and discovery relies on Brunet's DHT. Each VPN stores its P2P identifier into the DHT at the key generated by the desired IP address, such that the key, value pair is $(hash(IP), P2P)$. In order to ensure there are no conflicts, the storing of this value into the DHT uses an atomic operation, which succeeds only if no other peer has stored a value at $hash(IP)$.

The process for creating connections begins when IPOP receives an outgoing message. First it parses the destination address and queries the DHT for the remote peers P2P address. The peer then attempts to form a secure, direct connection with the remote peer using Brunet's secure messaging layer. Once that has formed, packets to virtual IP address are directed over that secure link.

In our original design [36], the virtual network was secured through a kernel-level IPsec stack, a model kept through our first generation Archer deployment. This approach limits security to virtual network links between parties leaving the P2P layer insecure; furthermore, in IPsec configuration each peer requires a unique rule for every other peer, which limited the maximum number of peers in the VPN. Without security for the P2P layer, malicious users could easily derail the entire system, but securing with IPsec would practically negate the benefits of the P2P system, because of network configuration issues related to NATs and firewalls. In our modern deployments, we have employed the security layer at the P2P layer, which in turn also secures virtual networking links.

Grids that rely upon VPNs to connect resources and users may impose the need for a certificate for the VPN and one for the grid. Our approach avoids this problem by using a VPN that allows a user to verify the identity of a remote peer and obtain its certificate, which then through software hooks in the grid software verifies the remote peers identity in terms relevant to the grid. In other words, user access is limited by the VPN and identity inside the grid is maintained by that same certificate. This might not be possible if all users were submitting from the same resources but is feasible in our system since each user submits from their own system.

2.3 Virtual Machines in Grid Computing

Earlier work [11] advocated the use of virtual machines (VMs) in grid computing for improved security and customization. Others since [29,19,4] have been established VMs as means for sandboxing, that is environments that allow untrusted users to use trusted resources in a limited fashion. VMs run as a process on a system, limiting processes running inside the VM from accessing the host operating system. Furthermore, a VM's network access may be limited by the host, effectively sealing them in a cage or sandbox protecting the hosts environment. VMs are also useful for customization and legacy applications, since a developer can configure the VM and then distribute it as an appliance, with the only requirement of the end user having VM manager software installed. Quantitatively, previous work has shown that CPU-bound tasks perform fairly well running with no more than 10% overhead and in some cases 0%, which is the case with VMs like Xen.

While not a direct correlation to grid computing, clouds have benefited significantly from VMs, as they provide the magic behind cloud infrastructures that provide IaaS, such as EC2. In these environments, users are able to create customized instances, or packaged operating systems and applications, inside of cloud environments; share with each other; and dynamically create or shutdown them as necessary. While the application of clouds is generic, it can easily be applied towards grids. A user can create push excess jobs into the cloud, when there is overflow, high demand, or the user lacks their own hardware infrastructure. One challenge, however, is the dynamic creation of a grid as well as extension of an existing grid using the cloud, challenges that are addressed in this paper.

3 Architectural Overview

Our approach attempts to reuse as many available components to design a grid middleware generic enough that the ideas can be applied to other middleware stacks. As a result, our contribution in this paper and in particular this section focuses primarily on the following key tasks: making grid construction easy, supporting decentralized user access, sandboxing the users environment, limiting access to the grid to authorized identities, and ensuring priority on users own resources.

3.1 Web Interface and the Community

Before deploying any software or configuring any hardware, a grid needs organization including certificate management, grid access, user account management, and delegation of responsibilities. These are complex requirements, which can be challenging to address, though for less restrictive systems, like a collection of academic labs sharing clusters, they may be very easy. One of the professors could handle the initial authorization of all the other labs and then delegate the responsibility of providing access to their their affiliates, such as students and scholars access.

For academic environments, grids become more challenging when the professor or worse yet students must maintain the certificates, handling certificate requests, and placing signed certificates in the correct location. Security is a serious topic and should ideally be handled by those that understand its nuances. Our solution to this potentially confusing area was a group interface, akin to something like Facebook's or Google's groups. Albeit, those types of groups are not hierarchal, which is a necessity in order to have delegated responsibilities. Thus we have a two layer approach, a grid group for members of the grid trusted by the grid organizers and user groups for those who are trusted by those in the grid group. Members of the grid group can create their own user groups. A member of a user group can gain access to the grid by downloading grid configuration data available within the user group web interface. This configuration data comes in the format of a disk image, when added to a "Grid Appliance" VM, it is used to obtain the user's credentials and enabling them to connect to the grid.

To give an example, consider our computer architecture grid, Archer. Archer was seeded initially by the University of Florida, so we are the founders and maintainers of the Archer grid group. As new universities and independent researchers have joined Archer, they request access to this group. Upon receiving approval, they then need to form their own user group so that they can allow others to connect to the grid. So a trusted member might create a user group titled "Archer for University X" and all members of university X will apply for membership in that group. The creator can make decisions to either accept or deny these users. Once the user has access, they will download their configuration data formatted as a virtual disk image and the "Grid Appliance" VM and start the "VM." After starting the VM, the user automatically be connected to Archer and able to submit and receive jobs.

Joining is easy; a grid requires a user to sign onto a website and download a configuration data, which can then be used on multiple systems. To support this process, the configuration data contains cryptographic information that facilitates acquisition of a signed certificate from the web interface through XML-RPC over HTTPS. The process begins by either booting the "Grid Appliance" or restarting a "Grid Appliance" service. When starting the service will detect if there is new configuration data, if there is, it contacts the web interface with the cryptographic information and a public key. The web interface verifies the user's identity using this cryptographic information, retrieves their profile from its database and binds that information with the public key to create a certificate request, which will then be signed and returned to the user.

With a public web interface, we have been able to create a variety communities. By bringing so many individuals from disperse locations, we have been able to foster a large distribution of peers who all contribute in a bootstrapping community for others in the P2P system on which the grids run. The web interface has been designed to support many grid groups, so too has the P2P infrastructure as it supports bootstrapping into unique private overlays for individual grids by means of Brunet's ability to support recursive bootstrapping. By using the public interface, users have an opportunity to reuse our bootstrap infrastructure and only need to focus on the configuration of their VPN and grid services, which has been trivialized to accepting or denying users access to a group and turning on resources. We would like to note that there is no need to make an explicit public grid community through the web interface, since all "Grid Appliances" come with a default configuration file that will connect them to an insecure public grid.

3.2 The Organization of the Grid

The previous section focused on facilitation of grid configuration using the web interface and skirted the issues of detailed configuration and organization. The configuration of the grid mirrors that of the connection pro-

	Description	Scalability	Job queue / submission site	API Requirements
Boinc	Volunteer computing, applications ship with Boinc and poll head node for data sets	Not explicitly mentioned, limited by the ability of the scheduler to handle the demands of the client	Each application has a different site, no separation from job queue and submission site	Applications are bundled with Boinc and must be written to use the Boinc API in order to retrieve data sets and submit results to the head node
BonjourGrid	Desktop grid, use zeroconf / Bonjour to find available resources in a LAN	No bounds tested, limits include multicasting overheads and processing power of job queue node	Each user has their own job queue / submission site	None
Condor	High throughput computing / on demand / desktop / etc / general grid computing	Over 10,000[1]	Global job queue, no limit on submission sites, submission site communicates directly with worker nodes	Optional API to support job migration and check pointing
PastryGrid	Use structured overlay Pastry to form decentralized grids	Decentralized, single node limited by its processing power, though collectively limited by the Pastry DHT	Each connected peer maintains its own job queue and submission site	None
PBS / Torque [26]	Traditional approach to dedicated grid computing	up to 20,000 CPUs[2]	Global job queue and submission site	None
SGE	Traditional approach to dedicated grid computing	Tested up to 63,000 cores on almost 4,000 hosts[3]	Global job queue and submission site	None
XtremWeb	Desktop grid, similar to Condor but uses pull instead of push, like Boinc	Not explicitly mentioned, limited by the ability of the scheduler to handle the demands of clients	Global job queue, separate submission site, optionally one per user	No built-in support for shared file systems

Table 1 Grid Middleware Comparison

cess. The first tier group maps to a common grid and each grid maps to a VPN. Thus when a user creates a new grid group, they are actually configuring a new VPN, which involves address range, security parameters, user agreements, and the name of the group. The system provides defaults for address range and security parameters, so users can focus on high level details like the user agreement and the grid's name.

As mentioned previously, the second tier of groups enables members in the grid group to provide access to their community. It is also the location that users download their configuration data. The configuration files come in three flavors: submission, worker, or manager. Worker nodes strictly run jobs. Submission nodes can run jobs as well as submit jobs into the grid. Manager nodes are akin to head nodes, those that manage the interaction between worker and submission nodes.

While the configuration details are handled by the web interface and scripts inside the "Grid Appliance," organization of the grid, more specifically the linking of worker and submission nodes to manager nodes, relies on the DHT. Managers store their IP addresses into the DHT at the key *managers*. When workers and clients join the grid, they automatically query this key, using the results to configure their grid software. Managers can also query this key to learn of other managers to coordinate with each other.

3.2.1 Selecting a Middleware

Our grid composition is largely based upon a desire to support a decentralized environment, while still retaining reliability and limiting our documentation support efforts. As there exist many middlewares to support job submission and scheduling, we surveyed available and established middleware to determine how well they matched our requirements. Our results are presented in Table 1, which covers most of the well established middleware and some recent research projects focused on decentralized organization.

Of the resource management middlewares surveyed, we chose to use Condor as it matches closest with our goals due to its decentralized properties and focus on desktop grids. With Condor, we are able to have multiple submission points, a non-trivial obstacle in some of the other systems. Additionally, adding and removing resources in Condor can be done without any configuration from the managers. Conversely, in SGE and Torque, after resources have been added into the sys-

[2] http://www.cs.wisc.edu/condor/CondorWeek2009/condor_presentations/sfiligoi-Condor_WAN_scalability.pdf

[3] http://www.clusterresources.com/docs/211

[4] http://www.sun.com/offers/docs/Extreme_Scalability_SGE.pdf

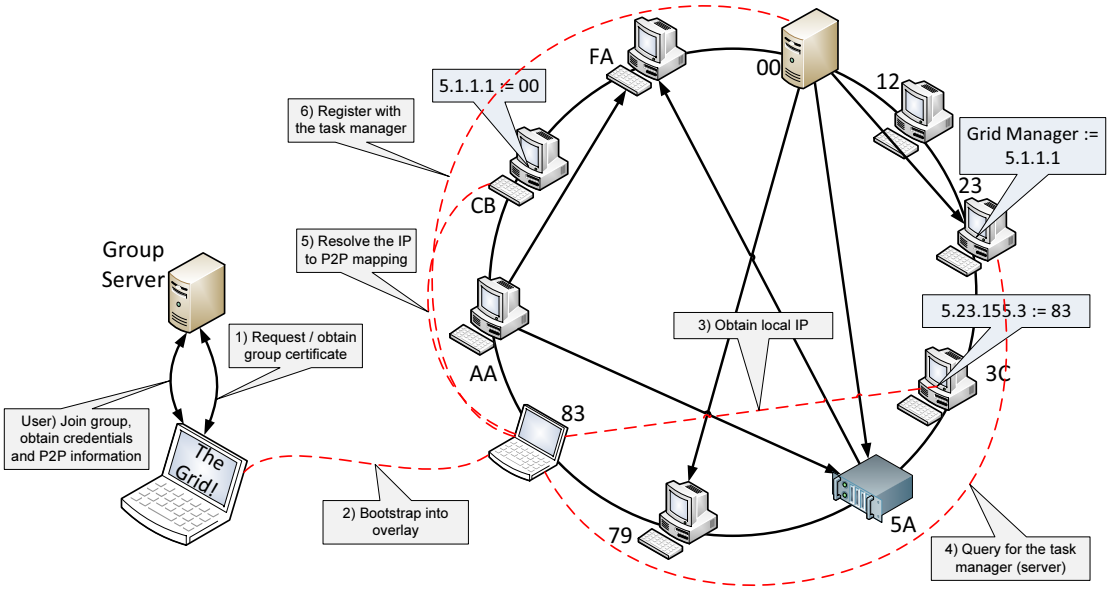

Fig. 2 An example deployment scenario: obtaining configuration files, starting the appliance, and connecting with a resource manager.

tem, the administrator must manually configure the manager to control them. Most scheduling software assumes that resources are dedicated, while Condor supports opportunistic cycles, by detecting the presence of other entities and will suspend, migrate, or terminate a job, thus enabling desktop grids. A common drawback to established middlewares is the requirement of a manager node; having no manager in an ad hoc grid would be ideal.

3.2.2 Self-Organizing Condor

While the requirement of a central manager may be undesirable, they can easily be run inside a VM and Condor supports the ability to run many in parallel through the use of "flocking [9]." Flocking allows submission sites to connect to multiple managers. This serves two purposes: 1) to provide transparent reliability by supporting multiple managers and 2) users can share their resources through their own manager. Flocking allows each site to run its own manager or share the common manager.

To configure Condor, manager IP addresses are stored into the DHT using the key *managers*. Joining peers query the DHT to obtain a list of managers, selecting one randomly to use as its primary manager with the result used for flocking. If the system prefers managers from its group, it will randomly contact each manager in an attempt to find a match, selecting one at random if no match is found. Until a manager is found, the process repeats every 60 seconds. Upon finding a manager, the state of the system is verified every 10 minutes and new managers are added to the flock list.

3.2.3 Putting It All Together

The following summarizes the configuration and organization of the grid. Minimally a grid will constitute a manager, some workers, and a submitter. Referencing Figure 2, a user begins by creating an account on the group server, joining a group, and obtaining configuration data for use with the "Grid Appliance." After adding the configuration data to a "Grid Appliance" system, the user will start the system initiating the automated system beginning with step "1." During which and without user interaction, each machine contacts the group website to obtain a valid VPN certificate. Whereupon, it connects to the P2P overlay whose bootstrap peers are listed inside the configuration file, "step 2." At which point, the machine starts the VPN service running on top of the P2P overlay, also part of step "2." The self-configuring VPN creates a transparent layer hiding from the user and administrators the complexity in setting up a common fabric that can handle potential network dynamics. Machines automatically obtain a unique IP address and find their place inside the grid, step "3." For a manager machine, this means registering in the DHT (not shown), while clients and workers search for available managers by querying the DHT, step "4;" IPOP translates the IP to a P2P address, step "5;" and then client contacts the manager directly, step "6."

3.3 Sandboxing Resources

As tasks can run on worker and potentially submission nodes, we have devised means to sandbox the environ-

ments that do not limit user interactions with the system. While more traditional approaches to sandboxing emphasize a separation between worker and submission machine, in our deployments, very few users explicitly deploy worker machines, most are submission machines. Thus we developed our sandboxing techniques to limit the ability of submitted jobs on systems that are simultaneously being used for submission. So our sandboxing technique considers more than just locking down the machine but also ensuring a reasonable level of access.

3.3.1 Securing the Resources

The core of our sandboxing approach is to limit attacks to software in the system and not poorly configured user space, such as poorly chosen passwords or resources external to the "Grid Appliance." All jobs are run as a set of predefined user identities. When the jobs are finished executing, whether forcibly shutdown or completed successfully, all processes from that user are shutdown, preventing malicious trojan attacks. Those users only have access to the working directory for the job and those with permission for everybody. Escalation of privilege attacks due to poor passwords are prevented by disallowing use of "su" or "sudo" for these users. Finally, through the use of firewalls, a running jobs network access is limited to the VPN, thus so long as jobs cannot escalate into root, they are unable to perform denial of service attacks on the Internet.

Additionally, systems can be configured such that the only network presented to them is that of the virtual network. To support this, IPOP has been enhanced to support a router mode, which can be bridged to a virtual machine adapter running on the host machine that connects to the network device running inside the VM. Not only does this improve performance, due to reduced I/O overhead, the same virtual network router can be used for multiple VMs.

To ensure that submit machines still have a high level of functionality without risking the system to external attacks even from users on the same network, user services are run only on a "host-only" network device within the virtual machine. This includes an SSH server and a Samba or Windows File Share. The user name and password matches that from the website. We would like to note that file sharing services work the opposite to that of host to guest as most VMs already have in place. Instead users can access their files on the VM from the host. This was done to limit potential attacks on submission machine.

3.3.2 Respecting the Host

Another aspect of sandboxing is respecting the usage of the host. While Condor can detect host usage on a machine it is running, when run inside a VM it cannot detect usage on the host. Thus it is imperative to support such a configuration otherwise our approach would be limited in that it can only be run during idle times. In the "Grid Appliance", this is addressed by running a light-weight agent on the host that communicates to the VM through the second Ethernet interface. The agent discovers a VM through multicast service discovery executed only on "host-only" virtual network devices. When a user accesses the host, the agent notifies a service in the VM, which results in running tasks being suspended, migrated, or terminated. The machine remains off limits until there has been no user activity for 10 minutes.

3.3.3 Decentralized Submission of Jobs

From the administrator's perspective, not requiring a submission machine is also a form of sandboxing. Maintaining a worker machine requires very low overhead, since jobs and their associated files are removed upon the completion of a job and corrupted workers can be deleted and redeployed. Maintaining a submission machine means user accounts, network access, providing data storage, and trusting users to play nicely on a shared resource. So having users be able to submit from their own resources reduces the overhead in managing a grid. It does come with a consequence, most grids provide shared file systems, which are statically mounted in all nodes. In a dynamic grid that might have multiple shares, this type of approach may not be very feasible.

All is not lost, for example, Condor provides data distribution mechanisms for submitted jobs. This can be an inconvenience, however, if only a portion of the file is necessary, as the entire file must be distributed to each worker. This can be particularly true with disk images used by computer architecture simulations and applications built with many modules or documentation. To support sparse data transfers and simplify access to local data, each "Grid Appliance" has a local NFS share exported with read-only permission. To address the issue of mounting a file system, there exists a tool to automatically mount file systems, autofs. autofs tool works by intercepting file system calls inside a specific directory, parsing the path, and mounting a remote file system. In the "Grid Appliance," accessing the path `/mnt/ganfs/hostname`, where hostname is either the IP address or hostname of an appliance, will automatically that appliance's NFS export without the need for

super-user intervention. Mounts are automatically unmounted after a sufficient period of time without any access to the mounted file system.

4 Deploying a Campus Grid

We now present a case study exploring a qualitative and quantitative comparison in deploying a campus grid and extending it into the "Cloud" using traditional techniques versus a grid constructed by "Grid Appliance." One of the target environments for the "Grid Appliance" is resources provided in distributed computer labs and many small distributed clusters on one or more university campus as shown in Figure 3. The goals in both these cases are to use commodity software, where available, and to provide a solution that is both simple but creates an adequate grid. In both cases, Condor is chosen as the middleware, which is a push scheduler and by default requires that all resources be on a common network thus a VPN will be utilized. Additionally, in this section, we cover details of the "Grid Appliance" that did not fit in the context of previous discussions in the paper.

4.1 Background

In this case study, we will compare and contrast the construction of two types of grids: a static grid configured by hand and a dynamic grid configured by the "Grid Appliance." Each grid is initially constructed using resources at the University of Florida and later extended to Amazon's EC2 and Future Grid at Indiana University using Eucalyptus. Each environment has a NAT limiting symmetric communication: University of Florida resources are behind two layers, first an "iptables" NAT and then a Cisco NAT; EC2 resources have a simple 1:1 NAT; and the Eucalyptus resources appear to have an "iptables" NAT.

4.2 Traditional Configuration of a Campus Grid

A VPN must be used to connect the resources due to the lack of network symmetry across the sites. There exists a wealth of VPNs available [23,42,30] and some explicitly for grids [18,34,33]. For simplicity sake, OpenVPN was chosen due to the simplicity in its configuration. In reality, OpenVPN makes a poor choice because it is centralized, thus all traffic between submitter and worker must traverse the VPNs server. Whereas others in the list are distributed and thus allow nodes to communicate directly, but in order to do so, manual setup is

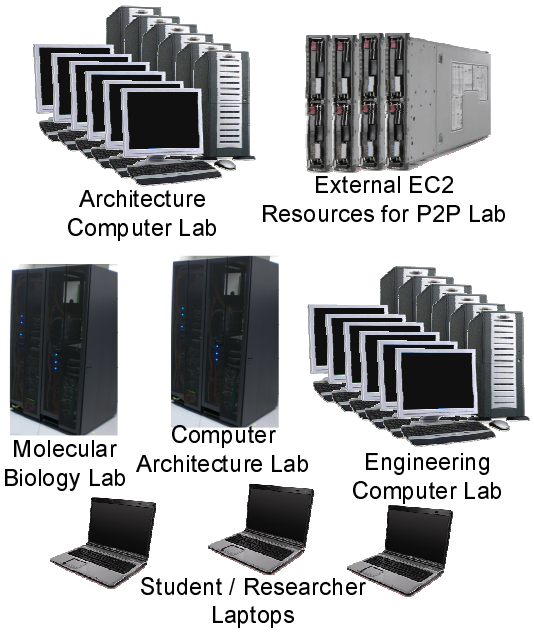

Fig. 3 A collection of various computing resources at a typical university.

required, a process, that would overwhelm many novice grid deployers. In all these cases, the VPN requires that at least a single node have a public address, thus we had to make a single concession in the design of this grid, that is, the OpenVPN server runs on a public node.

OpenVPN clients must be preconfigured with a signed certificate and knowledge of one ore more server's. While typically, most administrators would want a unique private key for each machine joining the grid, in our case study and evaluation, we avoided this process and used a common key, certificate pair. In doing so, there are potential dangers, for example, if any of the machines were hijacked, the certificate would have to be revoked and all machines would be rendered inoperable. To create a properly secured environment, each resource would have to generate or be provided a private key, a certificate request submitted to the certificate authority, and a signed certificate provided to the resource.

With the networking and security components in place, the next step is configuring grid middleware. Prior to deploying any resources, the manager must be allocated and its IP address provider to other resources in the system. Submission points are not a focus on this case study, though in general most systems of this nature have a single shared submission site. The challenges in supporting multiple submission points in this environment include creating certificates same as worker nodes, requiring users to configure OpenVPN and Condor, and handling NFS mounts. Whereas a single submission point creates more work for the system administrator, who will need to dedicate time for managing a resource with direct user interaction. Both approaches

have their associated costs and neither is trivial. The evaluation assumes a single user submitting from a single resource.

To address potential heterogeneity issues. An administrator would need to collaborate with others to ensure that all resources are running a common set of tools and libraries. Otherwise an application that works well on one platform could cause a segmentation fault on another, through no fault of the user, but rather due to library incompatibilities.

To export this system into various clouds, an administrator starts by running an instance that contains their desired Linux distribution and then installing the grid utilities like Condor and OpenVPN. Supporting individualization of the resources is challenging. The simplest approach is to store all the configuration in that instance including the single private key, certificate pair as well as the IP address of the manager node. Alternatively, the administrator could build an infrastructure that receives certificate requests and returns a certificate. The IP address of the manager node and of the certificate request handler could be provided to the cloud via user data, a feature common to most IaaS clouds that allows users to provide either text or binary data that is available via a private URL inside a cloud instance.

4.3 Grid Appliance in a Campus Grid

All these configuration issues are exactly the reasons why "Grid Appliance" and its associated group Web interface are desirable for small and medium scale grids. The first component is deciding which web interface to use, public (`www.grid-appliance.org`) or private hosted on their own resources. Similarly, users can deploy their own P2P overlay or use our shared overlay.

The web interface enforces unique names for both the users and the groups. Once the user has membership in the second tier of groups, they can download a file that will be used to automatically configure their resources. As mentioned earlier, this handled obtaining a unique signed certificate, connecting to the VPN, and discovering the manager in the grid. Configuration of the VPN and grid are handled seamlessly, the VPN automatically establishes direct links with peers on demand and peers configure based upon information available in the P2P overlay dynamically allowing for configuration changes.

Heterogeneity is a problem that will always exist if individuals are given governance of their own resources. Rather than fight that process, the "Grid Appliance" approach is to provide a reference system and then include that version and additional programs in the resource description exported by Condor. Thus a user looking for a specific application, library, or computer architecture can specify that in their job description. Additionally, by means of the transparent NFS mounts, users can easily compile their own applications and libraries and export them to remote worker nodes.

Extending the "Grid Appliance" system into the clouds is easy. The similarity between a VM appliance and a cloud instance are striking. The only difference from the perspective of the "Grid Appliance" system is where to check for configuration data. Once a user has created a "Grid Appliance" in a cloud, everyone else can reuse it and just supply their configuration data as the user data during the instantiation of the instances. As we describe in Section 6.2, creating "Grid Appliance" from scratch is a trivial procedure.

As described in detail earlier, an administrator needs to install the necessary software either by deploying VMMs and VM appliances or installing "Grid Appliance" packages on Debian / Ubuntu systems. Additionally, these systems need to be packaged with the configuration files or floppy disk images. At which point, the systems will automatically configure and connect to the grid. An administrator can verify this by monitoring Condor. Additional resources can be added seamlessly, likewise resources can be removed by shutting them off without direct interaction with the "Grid Appliance" or manager node.

4.4 Comparing the User Experience

In the case of a traditional grid, most users will contact the administrator and make a request for an account. Upon receiving confirmation, the user will have the ability to SSH into a submission site. Their connectivity to the system is instantaneous, their jobs will begin executing as soon as it is their turn in the queue. User's will most likely have access to a global NFS. From the user's perspective, the traditional approach is very easy and straightforward.

With the "Grid Appliance," a user will obtain an account at the web interface, download a VM and a configuration file, and start the VM. Upon booting, the user will be able to submit and receive jobs. To access the grid, users can either SSH into the machine or use the consoles in the VM. While there is no single, global NFS, each user has their own unique NFS and must make their job submission files contain their unique path. For the most part, the user's perspective of the "Grid Appliance" approach has much of the same feel as the traditional approach. Although users have additional features such as accessing their files via Samba

and having a portable environment for doing their software development.

4.5 Quantifying the Experience

For evaluation, we consider two epochs: time to bootstrap a grid assuming all user configuration has been completed and time to submit and complete a job on all available resources. The first task focuses on administrative time overheads for the self-configuring aspects of both systems, whereas the latter considers overheads introduced from the perspective of a user.

All configurations consisted of a single manager and submission node located at the University of Florida though on different resources. Because OpenVPN requires a public IP address, we ran a dedicated OpenVPN server also at the University of Florida. The "Grid Appliance" relies on the PlanetLab bootstrap system distributed across the world and a group server running at the University of Florida. Workers were started using virtual machines and cloud instances at Yale University using KVM and University of Chicago and the San Diego Supercomputer Center both using Nimbus as part of FutureGrid. 50 workers were started at each site and verified online prior to starting the experiment.

To isolate the cost of self-configuration, all machines booted into a grid free running state remaining in that state while polling a server for the initiation of the experiment. Workers query the server every 30 seconds waiting to find when to start the experiment. Upon initiating the experiment, the server would count down from 120 seconds with the workers being told amount of time they needed to wait prior to enable a simultaneous start. To determine how long it took for a node to connect to the manager, we examined the time the node appeared in the Condor Collector log and subtracted the initiation time as well as the 120 seconds delay.

The purpose of the job was to evaluate how long it took an independent client to connect to each resource in the grid, transfer a trivial sized file, and receive notification of job completion. Thus the job chosen was to sleep for 600 seconds.

The evaluation herein exhibits a significant improvement over our previous evaluation [37]. In this evaluation, we are not only considering OpenVPN and "Grid Appliance," we have recently done some radical redesign to the underlying systems to support a more active and event oriented system, to differentiate we call the previous system *old* and the current system *new*. Furthermore, our previous experiment had a significant deficiency with regards to determining the overhead of bootstrapping, as the time for resources to actually start could impact the time to get connected to the overlay as there was no separation between starting a resource and it connecting to the grid. Finally, we have included a set of resources that are on more constrained resources than before. Previously, resources either ran on a public IP address or all behind a common NAT. In this experiment, we have a set of workers behind a KVM NAT at Yale and the manager and submit node behind an IPTables firewall at the University of Florida.

The times measured include the time from when the last grid resource was started to the time it reported to the manager node, Figure 4, as well as the time required for the submit node to queue and run a 5 minute job on all the connected workers, Figure 5. The purpose of the second test is to measure the time it takes for a submission site to queue a task to all workers, connect to the workers, submit the job, and to receive the results; thus a stress test on the VPN's ability to dynamically create links and verifying all-to-all connectivity. The tests were run on 50 resources (virtual machines / cloud instances) in each environment and then on a grid consisting of all 150 resources with 50 at each site.

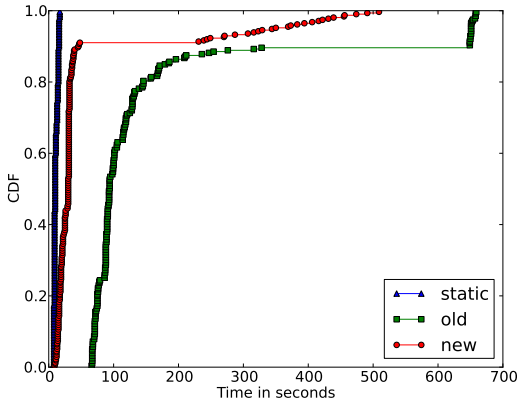

Fig. 4 Time in seconds for the self-configuring components of the grid to activate and connect workers to a manager.

Connection times in the grids are presented in Figure 4. Static or OpenVPN based grids certainly have the least amount of time in self-configuration as the system comes nearly completely configured. The only component left unconfigured is the IP address, which is provided by the central server. The old system relied on polling the VPN system to determine when an IP address was allocated in order to start Condor, in contrast to the new system which uses routing sockets. Additionally the new system contains significantly more aggressive NAT traversal and connection establishment. The large delay between connections in the old and new systems stems from the underlying polling nature of Condor. Condor uses a static delay in order

to determine how frequently to contact a manager. If the packet is dropped, the system will have to wait for the next cycle. Packet drops can occur in the P2P system, because of delayed connection establishment and potentially due to having an inadequate buffer in the VPN, which is part of an ongoing investigation. The slope in the curves relates to either the manager queue for more narrow slopes and to delays in connecting to the overlay for wider slopes.

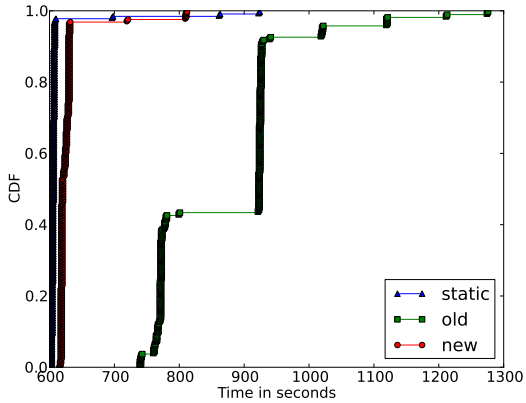

Fig. 5 Time in seconds for a 600 minute job to complete on resources connected to a grid.

The results from running jobs are shown in Figure 5. Interestingly enough, in no run was a single resource able to gain access to all resources in the first attempt, represented by nodes finishing significantly later than the origin. In the previous experiments [37], this was not the case; however, since this experiment consists of a more restrictive system the results were not unexpected. The most important conclusion is that our new approach presents an actual negligible overhead even for extremely short jobs.

The outcome of the evaluations suggest that our new approach has taken away much of the overheads that had previously existed. This is particularly impressive when considering all the additional automated work that the "Grid Appliance" handles in comparison to all the manual labor required in order to create the static systems, something that can not be reasonably quantitatively evaluated. Also since the evaluation consists of only a single submit node and a negligible transfer overheads of using OpenVPN are not readily apparent. In a system with multiple submitters, the OpenVPN server will become both a bandwidth and latency bottleneck in the system as all data must pass through it, which can be avoided using IPOP.

4.6 Batch Scheduling Tasks

The previous evaluation focused on the self-configuring aspects of the "Grid Appliance," it did not showcase the value offered by having seamless extensions to a grid. With the "Grid Appliance," a user can take an existing cluster and connect with others for mutual sharing of resources, users' can potentially complete tasks faster if not equally as fast with their cluster alone. We evaluated this behavior by constructing two different types of systems: 1) a cluster consisting of 10 resources and 1 submit node and 2) a grid created by the "Grid Appliance" consisting of 4 clusters each with 10 resources each and a submit node co-located with one of the clusters. Taking these systems, we ran 40 jobs such that each resource in the first system ran 4 and each resource in the second ran one job.

The first system consisted of resources located at the University of Florida all connected to the same network switch. The second system spanned the University of Florida, Florida State University, Northeastern University, and University of Texas at Austin with the submit node located at the University of Florida, all resources were behind various Cisco and IPtables based firewalls and NATs. The task selected came from a virtual computing course taught at the University of Florida[5], specifically use Simics with a simulated cache to run a GCC check point for 500,000,000 cycles. The specific details are unimportant as the purpose of the evaluation was to investigate wide area overheads of using the system. Another important detail is that we only use Condor for scheduling the job, files are actually transferred via the NFS, thus only blocks required for running the checkpoint will be transferred.

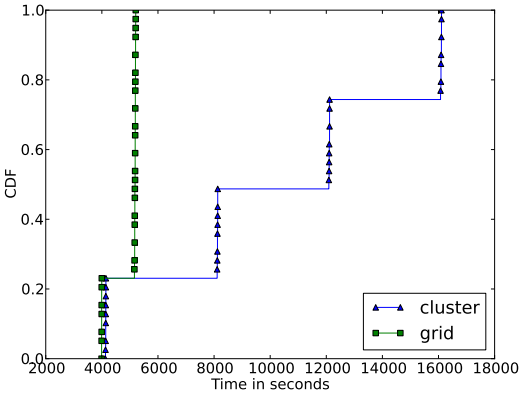

Fig. 6 Time in seconds for running batch tasks in a local cluster and a distributed grid.

[5] http://www.grid-appliance.org/wiki/index.php/EEL6892_Spring_11_HW_2:_Understanding_Simics

The results for our evaluation are presented in Figure 6. Clearly as anticipated, having more resources solved the problem much more quickly. The graph shows a completion delay between resources running locally, at the University of Florida, in comparison to those running at the other institutions. While network overheads certainly played a role in this, they were not as significant as the fact that the hardware at the other sites was one year older and clearly out performed by the University of Florida resources. The only reason why the first round of cluster results came in more slowly than the grid was due to Condor's polling based scheduling system.

5 An Academic Tool for Parallel Computing

Our second case study explores the possibility of using Grid Appliance as a medium for enabling academic investigations into parallel computing systems. While these frameworks are not particularly complicated in configuring, they may take considerable time for first time users. Also they require the user to have complete control over the set of resources. Using Condor to deploy these types of jobs requires that submission from a single entry point and resources be dedicated, something not typically done in Condor pools. As a result, we defined a container for parallel computing systems that works well with the two we have considered thus far: MPICH2[6] and Hadoop[7].

5.1 Traditional Configuration

Both MPICH2 and Hadoop require a similar setup. They both begin by an administrator installing the frameworks and the necessary support libraries on each resource. In order for the systems to *s*elf-configure, the administrator will need to install a common user and SSH key on each resource. All but one of the resources will be slave nodes with the other being the master node. The system is configured and tasks are submitted from the master node, which will push the requests out to the slaves. In order to connect to the slaves, the master keeps a file containing all the hostnames of the slaves and has access to the private SSH key enabling passwordless access to the slaves. At which point, the systems can be started and tasks submitted.

[6] `http://www.mcs.anl.gov/research/projects/mpich2/`
[7] `http://hadoop.apache.org`

5.2 Grid Appliance Configuration

In designing our framework, we wanted a user to go from an unconfigured environment to running tasks with as minimal effort as possible. Using the "Grid Appliance's" other components as our baseline, we already had all-to-all connectivity and the ability to start applications remotely via IPOP and Condor, respectively. Our initial approach was to use Condor to distribute configuration in order to run SSH server at each node and then use this SSH session in configuration of the parallel framework. Unfortunately, this approach was fraught with problems. Many of these tools assumed a common user on each resource; however, in Condor, a unique user will be used for each task in a multiple CPU machine. Additionally, there was a non-trivial issue of ensuring that multiple SSH session for parallel frameworks would not conflict.

Going back to the drawing boards, we dissected the parallel frameworks in order to understand how they worked. While the traditional configuration of most parallel frameworks follow what was described earlier, the use of a single point capable of starting and stopping the system via SSH is not required. By investigating the actual SSH calls, we discovered that the SSH was starting a binary on each of the slaves and passing in by command-line arguments information about the master. The rest of the configuration happened between processes running on the master and slaves independent of any channeling via SSH.

With this knowledge in hand, we began applying the constraints provided by the "Grid Appliance" to this configuration. The new approach was to start a Python based XML-RPC server on the submission node, it would be responsible for waiting for all nodes to join as well as triggering the start and end of tasks. Each of the slaves would start an XML-RPC server as well, register their IP and port with the master and wait for a notification to start. Once the master heard from all the slaves, it would notify the slaves to configure the parallel framework. At this point, the framework would be running and tasks could be submitted into the system. The system supports both interactive sessions as well as batch requests. At the conclusion of batch requests, the master will notify the slaves via XML-RPC and notify them to end. An interactive session would conclude with a user running a script that would notify the master's XML-RPC server to initiate termination with the slaves.

The only unresolved issue is dealing with frameworks files and required libraries, which may not be installed on the remote machines. Fortunately, each "Grid Appliance" has a NFS mount, which can be used to

host these files. When a job gets executed on a remote machine, it simply updates the appropriate Unix environmental variables to know about the NFS and the libraries and files it hosts.

The end result is the following. The user installs our package for MPI or Hadoop into their NFS share. The user will then run a script that requests how many nodes to provide and optionally a file for batch tasks. This will start the XML-RPC server and generate a condor submit script which will then be submitted to Condor. As the nodes are claimed by Condor for this task, the XML-RPC server on the master will receive incoming requests for registration. Once all the expected slaves have registered, the master will send a signal to all of them notifying them to start the parallel framework. Each slave will access the required files by the NFS share from the master and start the parallel framework. Upon completion, they will notify the master of the port where they started the parallel framework session. When all have responded, the master finalizes its configuration and either batch tasks will be executed or user interaction will be enabled.

5.3 Quantifying the Experience

6 Lessons Learned

This section highlights some the interesting developments and experiences, we have had that do not fit the topics discussed so far.

6.1 Deployments

A significant component of our experience stems from the computational grid provided by Archer [10], an active grid deployed for computer architecture research, which has been online for over 3 years. Archer currently spans six seed universities contributing over 600 CPUs as well as contributions and activities from external users. The Archer grid has been accessed by hundreds of students and researchers from over a dozen institutions submitting jobs totaling over 500,000 hours of job execution in the past two years alone. The batch scheduling of Simics evaluation actually used Archer resources. In practice, we have seen that there has been minimal conflict in resource demands on the syste, and even when there is higher demand, each user is guaranteed high priority on their own resources. Thus we have only heard praises and no complaints with regards to the availability of resources.

The Grid Appliance has also been utilized by groups at the Universities of Florida, Clemson, Arkansas, and Northwestern Switzerland as a tool for teaching grid computing. Meanwhile the universities of Clemson and Purdue are using the Grid Appliance's VPN (GroupVPN / IPOP) to create their own grid systems. Over time, there have been many private, small-scale systems using our shared system available at `www.grid-appliance.org` with other groups constructing their own independent systems. Feedback from users through surveys have shown that non-expert users are able to connect to our public Grid appliance pool in a matter of minutes by simply downloading and booting a plug-and-play VM image that is portable across VMware, VirtualBox, and KVM.

6.2 Towards Unvirtualized Environments

Because of the demands put on Archer in terms of avoiding the overheads of virtualization and the perceived simplicity of managing physical resources as opposed to virtual resources running on top of a physical resources, many users have requested the ability to run Grid Appliances directly on their machine. Unlike clouds with machine images such as AMIs or VM appliances, physical machines images cannot be easily exported. Most physical OS installed on physical machines will need some some custom tailoring to handle environment specific issues.

With this in mind, we moved away from stackable file systems and towards creating repositories with installable packages, such as DEB or RPM. The implications of packages mean that users can easily produce "Grid Appliances" from installed systems or during system installation. With the VPN router mode, mentioned earlier, resources in a LAN can communicate directly with each other rather than through the VPN. That means if they are on a gigabit network, they can full network speeds as opposed to being limited to 20% of that due to the VPN, overheads discussed in [39].

Using an APT repository in combination with unattended upgrades, we have enabled our Debian based systems to automatically upgrade minimizing concerns about upgrade failures and requirements for user intervention. This is in contrast to our old method of having a single monolithic package and package management technique that we employed previously. Using existing package managers and distribution systems reduces our dependency count and also enables users to easily accept or reject upgrades by following well documented procedures.

6.3 Advantages of the Cloud

We have had the experience of deploying the "Grid Appliance" on three different cloud stacks: Amazon's EC2 [2], Future Grid's Eucalyptus [25], and Future Grid's Nimbus [21]. All of the systems, encountered so far, allow for data to be uploaded with each cloud instance started. The instance can then download the data from a static URL only accessible from within the instance, for example, EC2 user data is accessible at `http://169.254.169.254/latest/user-data`. A "Grid Appliance" cloud instances can be configured via user-data, which is the same configuration data used as the virtual and physical machines, albeit zip compressed. The "Grid Appliance" seeks the configuration data by first checking for a physical floppy disk, then in specific directory (`/opt/grid_appliance/var/floppy.img`), followed by the EC2 / Eucalyptus URL, and finally the Nimbus URL. Upon finding a floppy and mounting it, the system continues on with configuration. Clouds have been also very useful for debugging. Though Amazon is not free, with Future Grid, grid researchers now have free access to both Eucalyptus and Nimbus clouds. Many bugs can be difficult to reproduce in small system tests or booting one system at a time. By starting many instances simultaneously, we have been able to quickly reproduce and isolate problems, leading to timely resolutions, and verification of those fixes.

6.4 Stacked File Systems

Configuring systems can be difficult, which makes it important to have the ability to share the resulting system with others. The approach of actually creating packages can be overly complicated for novices. To address this concern, our original "Grid Appliance" supported a built-in mechanism to create packages through a stackable file system using copy-on-write, as describe in [36]. In this environment, the VM used 3 disks: the "Grid Appliance" base image, the software stack configured by us; a module; and a home disk. In normal usage, both the base and module images are treated as read-only file systems with all user changes to the system being recorded by the home image, as depicted in Figure 7.

To upgrade the system, users replaced their current base image with a newer one, while keeping their module and home disks. While the purpose of the module was to allow users to extend the configuration of the "Grid Appliance." To configure a module the system would be booted into developer mode, an option during the boot phase, where only the base and module images are included in the stacked file system. Upon completing the changes, a user would run a script that would clean the system and prepare it for sharing. A user could then share the resulting module image with others.

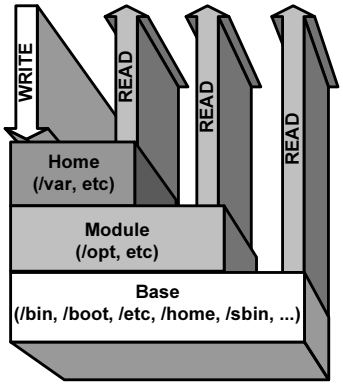

Fig. 7 Example of a stackable file system from our previous "Grid Appliance." A file will be read from the top most file system in the stack and all writes are directed to Home.

Issues with this approach made it unattractive to continue using. First, there exists no kernel level support for stackable file systems, we had to add UnionFS [41] to the kernel, adding the weight of maintaining a kernel upon our shoulders. While FUSE (filesystem in userspace) solutions exist, they require modifications to the initial ram disk, which is reproduced automatically during the installation of every new kernel, furthermore, our experience with them suggests they are not well suited for production systems. Additionally, the approach was not portable to clouds or physical resources. So while we have deprecated the feature for now, we see it as a potential means to easily develop packages like DEB and RPM.

6.5 Priority in Owned Resources

In Archer, seed universities should have priority on the resources at their university. Similarly, users should have priority on their contributions. Otherwise, users will remove their resources from the grid, when they want guaranteed access. To support user and group based priorities, Condor has mechanisms that can be enforced at the server that allow for arbitrary means to specify user priority for a specific resource. So our configuration specifies that if the resource's user or group matches that of the submitter, the priority is higher than otherwise. This alone is not sufficient as malicious users could easily tweak their user name or group to obtain priority on all resources. Thus whenever this check is made the user's identity in the submission information is verified against their P2P VPN certificate. Failed matches are not scheduled and are stored in a log at the manager for the administrator to deal with later.

6.6 Timing in Virtual Machines

Certain applications, particularly license servers, are sensitive to time. Because of the nature of grids, there exist possibilities of having uncoordinated timing, such as improperly specifying the time zone or not using a network time protocol (NTP) server With regards to VMs, VMWare [35] suggests synchronizing with the host's time and to avoid using services like NTP, which may have adverse affects on timing inside the virtual machine. While NTP might have some strange behavior, relying on host time may produce erratic jumps in time that some software cannot handle. Our experiences recommends the use of NTP to address these concerns, which has resolved many issues with strange software behavior and frustration from users when their jobs fail due to being unable to obtain a license due to a timing mismatch.

6.7 Selecting a VPN IP Address Range

One challenge in deploying a VPN is ensuring that the address space does not overlap with that over the environments where it will be used. If there is overlap, users will be unable to connect to the VPN. Doing so will confuse the network stack, as there will be two network interfaces connected to the same address space but different networks. A guaranteed, though not necessarily practical solution is to run the resource on a VM NAT or a cluster NAT that does not overlap the IP address space of the VPN.

Users of the "Grid Appliance" should not have to concern themselves with this issues. Prior work on the topic by Ala Rezmerita et al. [27] recommends using the experimental address class E ranging between 240.0.0.0 - 255.255.255.254, unfortunately this requires Linux kernel modifications. With the amount of bugs and security fixes regularly pushed into the kernel, maintaining a forked kernel requires a significant amount of time, duplicating the work already being performed by the OS distribution maintainers. This would also limit the ability to easily deploy resources in physical and cloud environments. Additionally, users that wanted to multipurpose a physical resource may not want to run a modified kernel, while in most cloud setups the kernel choice is limited.

We have since moved towards using the 5.0.0.0 - 5.255.255.255 address range. Like the class E address space it is unallocated, but it requires no changes to any operating systems. The only limitation is that some other VPNs also use it, thus a user would not be able to run two VPNs on the same address space concurrently. This approach is much better than providing kernels or dealing with network address overlaps. Interestingly, even with this in place, we still see some "GroupVPNs" using address ranges in normal private network address ranges for the VPN, like 10.0.0.0 - 10.255.255.255 and 192.168.0.0 - 192.168.255.255.

6.8 Administrator Backdoor

While most administrators will agree that most problems that users encounter are self-inflicted, there are times, when the system is at fault. Debugging systems faults in a decentralized system can be very tricky, since it is very difficult to track down a resource in order to gain direct physical access. Additionally, having a user bring their resource to an administrator may be prohibitively complicated, as the user would need to relocate their "Grid Appliance" instance and have network connectivity in order to connect to the grid and show the problem to the administrator. To address this and other concerns that only appear after running the system for long periods of time, we have supplied an administrator backdoor into all resources by installing our public ssh key, though users are informed of this and are free to remove it for privacy concerns. In typical configurations, this approach might not be feasible, but because the "Grid Appliance" ships with a decentralized VPN supporting all-to-all connectivity, any resource connected to the VPN is accessible for remote debugging by an administrator. Most users involved are extremely delighted with the process as it has an appearance that the system "just works."

6.9 Enabling Access to Manager and Worker Systems

In the original system, we used the password "password" as the default for directly interacting with a machine. Since this is inherently insecure, we disable "su" and "sudo," but also only enabled accounts on submit nodes, where users would be guaranteed to access the system and hence more likely to change the password. Unfortunately this prevents users from easily accessing worker and manager resources. More advanced users who were setting up their own systems would occasionally want to access these resources for debugging purposes or to enable additional features in live systems. While a user could provide a SSH key in the floppy, which is done for the administrator back door, there is not a clear cut way for a user to identify a resource without directly accessing it via a console. We have devised some multicast discovery applications as well as employed network search tools such as nmap, but their use is not trivial. By using the password as stored in

the database, we can enable accounts on worker and manager resources with minimal concern of hijacks assuming of course that the user properly selected their password at account creation.

Implementation of this feature was not straight forward. Most web sites do not store passwords in the same format as the Linux password database, and converting between two one-way functions is mathematically challenging. As a result, we needed to convert the password at the time of user account creation. Because we were adding this to an existing system, we also needed to take into account existing users, and so we wrote a hook that would obtain the password during authentication and store the Linux formatted password to the database. During the creation of a user account on a "Grid Appliance," the encrypted format of the users password is then passed as a parameter into the user account adding application, *adduser*. While for our system, we experienced no show stoppers, a system incorporating javascript and XML-RPC (AJAX) to perform authentication may.

7 Related Work

Existing work that falls under the general area of desktop grids/opportunistic computing include Boinc [3], BonjourGrid [1], and PVC [27]. Boinc, used by many "@home" solutions, focuses on adding execute nodes easy; however, job submission and management rely on centralization and all tasks must use the Boinc APIs. BonjourGrid removes the need for centralization through the use of multicast resource discovery; the need for which limits its applicability to local area networks. PVC enables distributed, wide-area systems with decentralized job submission and execution through the use of VPNs, but relies on centralized VPN and resource management.

Each approach addresses a unique challenge in grid computing, but none addresses the challenge presented as a whole: easily constructing distributed, cross-domain grids. Challenges that we consider in the design of our system include allowing submission sites to exist any where without being confined to complex configuration or highly available, centralized locations; the ability to dynamically add and remove resources by starting and stopping a a resource; and the sharing of common servers so that no group in the grid is dependent on another. We emphasize these points, while still retaining the ease of use of Boinc, the connectivity of PVC, and the flexibility of BonjourGrid. The end result is a system similar to OurGrid [5]; however, OurGrid requires manual configuration of the grid and networking amongst sites, administration of users within a site, and limits network connectivity amongst resources, whereas "Grid Appliance" transparently handles these issues with a P2P overlay and VPN to handle network constraints and support network sandboxing and a web interface to configure and manage the grid.

With regards to clouds, there exists contextualization [20]. Users construct an XML configuration file that describes how a cloud instance should be configured and provide this to a broker. During booting of a cloud instance, it will contact a third-party contextualization broker to receive this file and configure the system. This approach has been leveraged to create dynamic grids inside the Nimbus cloud [17]. While this approach can reproduce similar features of the "Grid Appliance," such as creating grids inside the cloud, there are challenges in addressing cloud bursting, automated signing of certificates, and collaboration amongst disparate groups.

Contextualization has made even further in roads through a project known as "Globus Provision[8]," which creates an entire Globus and Condor pool in the clouds. Our approaches are orthogonal in many ways. For example, Globus seeks to create a traditional grid with users sharing common submission points and having dedicated entry points into and across the grid. While "Globus Provision" could allow remote access into a "Grid Appliane" system, it is not trivial to add a "Grid Appliance" system into a "Globus Provision" system. As the "Globus Provision" system would require each user to authenticate with the "Globus" agents or have a set of trusted systems act on behalf of the "Grid Appliance" system.

8 Conclusions

In this paper, we have presented a grid framework that enables users to easily deploy their own grids. By reducing the entry barrier to constructing wide-area grids, rather than just providing a grid, our approach teaches users to create grids rather than providing access. The features of the "Grid Appliance" significantly simplify the construction of a grid over traditional approaches. These methods are based upon and have been verified through experience with individuals and groups coming from various backgrounds. Furthermore, we have presented both qualitative and quantitative utility of the "Grid Appliance" in Section 4. Namely, decentralized, P2P VPNs are resilient and easily configured; web interfaces ease the burden of crafting configuration files and signing of certificates; and package management systems can be used to create appliances nearly as con-

[8] `http://globus.org/provision`

veniently as VMs. Those interested are able to test drive the system by coming to our public Web interface at the `www.grid-appliance.org`, where they can either use our public testing grid or deploy their own.

The concepts in this paper are intentionally generic so that they can easily be applied to other systems. For example, more complex approaches to grids involve entities known as virtual organizations. A virtual organization allows the same set of resources to be members of many distinct grids. Our web interface idea could be extended to support virtual organizations. Additionally, the sandboxing technique could be applied to many environments, including OurGrid, to allow grid network access without compromising the safety of the system.

For future work, we are considering mechanisms to fully decentralize the "Grid Appliance" by using a decentralized grid system that requires no manager nodes, though the challenges in doing so, are efficient resource discovery, clustering of group resources, and fair use scheduling. A completely decentralized grid could be constructed completely by client machines, in which, no one is more responsible than another for maintaining the grid. While the parallel framework works quite well it can still be significantly improved by attempting to be scheduled on resources based upon proximity and supporting some form of reliability in order to deal with disappearing or reclaimed nodes.

Acknowledgments

This work is sponsored by the National Science Foundation (NSF) under awards 0751112 and 0721867. This material is based upon work supported in part by the NSF under grant 091812 (Future Grid). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

1. Abbes, H., Cérin, C., Jemni, M.: Bonjourgrid: Orchestration of multi-instances of grid middlewares on institutional desktop grids. In: International Parallel and Distributed Processing Symposium (IPDPS) (2009)
2. Amazon.com, Inc: Amazon elastic compute cloud. `http://aws.amazon.com/ec2` (2009)
3. Anderson, D.P.: Boinc: A system for public-resource computing and storage. In: the International Workshop on Grid Computing (2004)
4. Andrade, N., Costa, L., Germoglio, G., Cirne, W.: Peer-to-peer grid computing with the ourgrid community. In: Brazilian Symposium on Computer Networks (2005)
5. Andrade, N., Costa, L., Germglio, G., Cirne, W.: Peer-to-peer grid computing with the ourgrid community. In: Brazilian Symposium on Computer Networks (SBRC) - 4th Special Tools Session (2005)
6. Andreetto, P., Andreozzi, S., Avellino, G., Beco, S., Cavallini, A., Cecchi, M., Ciaschini, V., Dorise, A., Giacomini, F., Gianelle, A., Grandinetti, U., Guarise, A., Krop, A., Lops, R., Maraschini, A., Martelli, V., Marzolla, M., Mezzadri, M., Molinari, E., Monforte, S., Pacini, F., Pappalardo, M., Parrini, A., Patania, G., Petronzio, L., Piro, R., Porciani, M., Prelz, F., Rebatto, D., Ronchieri, E., Sgaravatto, M., Venturi, V., Zangrando, L.: The glite workload management system. Journal of Physics: Conference Series **119**(6), 062,007 (2008)
7. Boykin, P.O., Bridgewater, J.S.A., Kong, J.S., Lozev, K.M., Rezaei, B.A., Roychowdhury, V.P.: A symphony conducted by brunet. `http://arxiv.org/abs/0709.4048` (2007)
8. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: Symposium on Operating Systems Principles (SOSP). ACM, New York, NY, USA (2007)
9. Epema, D.H.J., Livny, M., van Dantzig, R., Evers, X., Pruyne, J.: A worldwide flock of condors: Load sharing among workstation clusters. Future Generation Computer Systems **12**(1), 53 – 65 (1996)
10. Figueiredo, R.J., Boykin, P.O., Fortes, J.A.B., Li, T., Peir, J., Wolinsky, D., John, L.K., Kaeli, D.R., Lilja, D.J., McKee, S.A., Memik, G., Roy, A., Tyson, G.S.: Archer: A community distributed computing infrastructure for computer architecture research and education. In: CollaborateCom (2008)
11. Figueiredo, R.J., Dinda, P.A., Fortes, J.A.B.: A case for grid computing on virtual machines. In: International Conference on Distributed Computing Systems. IEEE Computer Society (2003)
12. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. Journal of Computer Science and Technology **21**, 513–520 (2006). URL `http://dx.doi.org/10.1007/s11390-006-0513-y`. 10.1007/s11390-006-0513-y
13. Ganguly, A., Agrawal, A., Boykin, O.P., Figueiredo, R.: IP over P2P: Enabling self-configuring virtual IP networks for grid computing. In: International Parallel and Distributed Processing Symposium (2006)
14. Ganguly, A., Agrawal, A., Boykin, P.O., Figueiredo, R.: Wow: Self-organizing wide area overlay networks of virtual workstations. In: IEEE High Performance Distributed Computing (HPDC) (2006)
15. Ganguly, A., Boykin, P.O., Wolinsky, D., Figueiredo, R.J.: Improving peer connectivity in wide-area overlays of virtual workstations. Cluster Computing Journal (2009)
16. Ganguly, A., Wolinsky, D., Boykin, P., Figueiredo, R.: Decentralized dynamic host configuration in wide-area overlays of virtual workstations. In: International Parallel and Distributed Processing Symposium (2007)
17. Harutyunyan, A., Buncic, P., Freeman, T., Keahey, K.: Dynamic virtual AliEn grid sites on nimbus with CernVM. Journal of Physics: Conference Series (2010)
18. Jiang, X., Xu, D.: Violin: Virtual internetworking on overlay. In: International Symposium on Parallel and Distributed Processing and Applications, pp. 937–946 (2003)
19. Keahey, K., Doering, K., Foster, I.: From sandbox to playground: Dynamic virtual environments in the grid. In: International Workshop in Grid Computing (2004)
20. Keahey, K., Freeman, T.: Contextualization: Providing one-click virtual clusters. In: eScience (2008)
21. Keahey, K., Freeman, T.: Science clouds: Early experiences in cloud computing for scientific applications. In: Cloud Computing and Its Applications (2008)

22. Livny, M., Basney, J., Raman, R., Tannenbaum, T.: Mechanisms for high throughput computing. SPEEDUP Journal **11**(1) (1997)
23. LogMeIn: Hamachi. `https://secure.logmein.com/products/hamachi2/` (2009)
24. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: International Workshop on Peer-to-Peer Systems (2002)
25. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid) (2009)
26. Resources, C.: Torque resource manager. `http://www.clusterresources.com/pages/products/torque-resource-manager.php` (2007)
27. Rezmerita, A., Morlier, T., Neri, V., Cappello, F.: Private virtual cluster: Infrastructure and protocol for instant grids. In: Euro-Par (2006)
28. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) (2001)
29. Santhanam, S., Elango, P., Dusseau, A.A., Livny, M.: Deploying virtual machines as sandboxes for the grid. In: WORLDS (2005)
30. Sliepen, G.: tinc. `http://www.tinc-vpn.org/` (2009)
31. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking **11**(1) (2003)
32. Sun: gridengine. `http://gridengine.sunsource.net/` (2007)
33. Sundararaj, A.I., Dinda, P.A.: Towards virtual networks for virtual machine grid computing. In: Conference on Virtual Machine Research And Technology Symposium, pp. 14–14 (2004)
34. Tsugawa, M., Fortes, J.: A virtual network (vine) architecture for grid computing. International Parallel and Distributed Processing Symposium (2006)
35. VMware, Inc.: Timekeeping in vmware virtual machines. `http://www.vmware.com/pdf/vmware_timekeeping.pdf` (2008)
36. Wolinsky, D.I., Agrawal, A., Boykin, P.O., Davis, J., Ganguly, A., Paramygin, V., Sheng, P., Figueiredo, R.J.: On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations. In: International Workshop on Virtualization Technologies in Distributed Computing (2006)
37. Wolinsky, D.I., Figueiredo, R.: Experiences with self-organizing decentralized grids using the grid appliance. In: International Symposium on High Performance Distributed Computing (ACM HPDC 2011) (2011)
38. Wolinsky, D.I., Lee, K., Boykin, P.O., Figueiredo, R.: On the design of autonomic, decentralized vpns. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing (2010)
39. Wolinsky, D.I., Liu, Y., Juste, P.S., Venkatasubramanian, G., Figueiredo, R.: On the design of scalable, self-configuring virtual networks. In: IEEE/ACM Supercomputing 2009 (2009)
40. Wolinsky, D.I., St. Juste, P., Boykin, P.O., Figueiredo, R.: Addressing the P2P bootstrap problem for small overlay networks. In: 10th IEEE International Conference on Peer-to-Peer Computing (P2P) (2010)
41. Wright, C.P., Zadok, E.: Unionfs: Bringing file systems together. In: Linux Journal (2004)
42. Yonan, J.: OpenVPN. `http://openvpn.net/` (2009)