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

David Isaac Wolinsky, Arjun Prakash, Renato Figueiredo
University of Florida

***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, potentially long job queues, and environments and policies that might affect a user’s work flow. In many scenarios, individual users and communities can benefit from less restrictive, dynamic grids with flexibility in deploying resources, sharing access, and configuring the environment, even if this results in smaller scale systems.

In this paper, we address how small groups can dynamically create and join grid infrastructures with emphasis on low administrative overhead. Our work distinguishes itself from other projects with similar goals in three main ways: decentralized system organization, decentralized user access, and web interfaces to construct grids. This paper presents the “Grid Appliance,” originally designed to create a WOW or wide area overlay network of virtual workstations that has developed over the past 6 years into a mature system with many users. Beyond describing the architecture of this system, this paper contains lessons learned during the development from abstract WOW to a complete user-friendly system and a case study backed by quantitative analysis that verifies the utility of our approach.

I. Introduction

Grid computing presents opportunities to combine distributed resources to form powerful systems. Due to the challenges in coordinating the organization of grids, many researchers become members of existing grids or deploy their own private resources. The limitations of grids in terms of flexibility and policies may push potential users into creating and managing their own computer clusters. For private systems, there exists a wealth of middleware available, including resource managers such as Condor [1], Torque (PBS) [2], and Sun Grid Engine [3], though many see the entry barrier to installing and future management of these systems as being greater than their usefulness and as a result turn to inefficient ad hoc resource discovery and allocation.

Local systems that have been successfully assembled with the appropriate middleware can be shared with other users working in the same field or in different departments or groups within the same organization to form even larger systems or grids. Some of the resource managers listed earlier support simple grids, while others turn to more complete solutions like the Globus Toolkit [4] or gLite [**?**]. 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 advent of cheap on-demand computing through the ”Cloud” by means of infrastructure-as-a-service, small computer clusters in time of need can be enlarged dynamically into much larger systems. Making this happen has similar demands to that of creating a distributed grid with emphasis on connectivity and organization. Cloud resources, like any other computer resource, needs to be configured, and while it could be configured by a third-party for grid computing, it will stilll need some configuration from the deployer in terms of allocating the appropriate certificates and potential network configuration.

In the cases presented, a user with a sufficient desire or knowledge in cyberinfrastructure could achieve the desired environment. In this paper, we present techniques that enable those less motivated or who lack the necessary expertise to deploy and extend ad hoc, distributed grids. To verify this assertion, we have implemented a system supporting these ideas in the “Grid Appliances,” which as will be demonstrated, allows users to focus on making use of the grids 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.

The “Grid Appliance” project and concepts have been actively developed and used in several projects for the past 6 years. Of these projects, Archer, a distributed grid for computer architecture research spanning 6 seed universities, stands out as the most ambitious but successful project. Researchers interested or desiring the ability to access both grid resources and specialized tools like Simics can easily use and contribute resources with little time overhead. Users join a website, download a configuration image and a virtual machine (VM), and start 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.

At the heart of our approach lies a P2P infrastructure based upon a distributed hash table (DHT) useful for decentralized configuration and organization of systems. 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 original purpose of the DHT was to support a decentralized

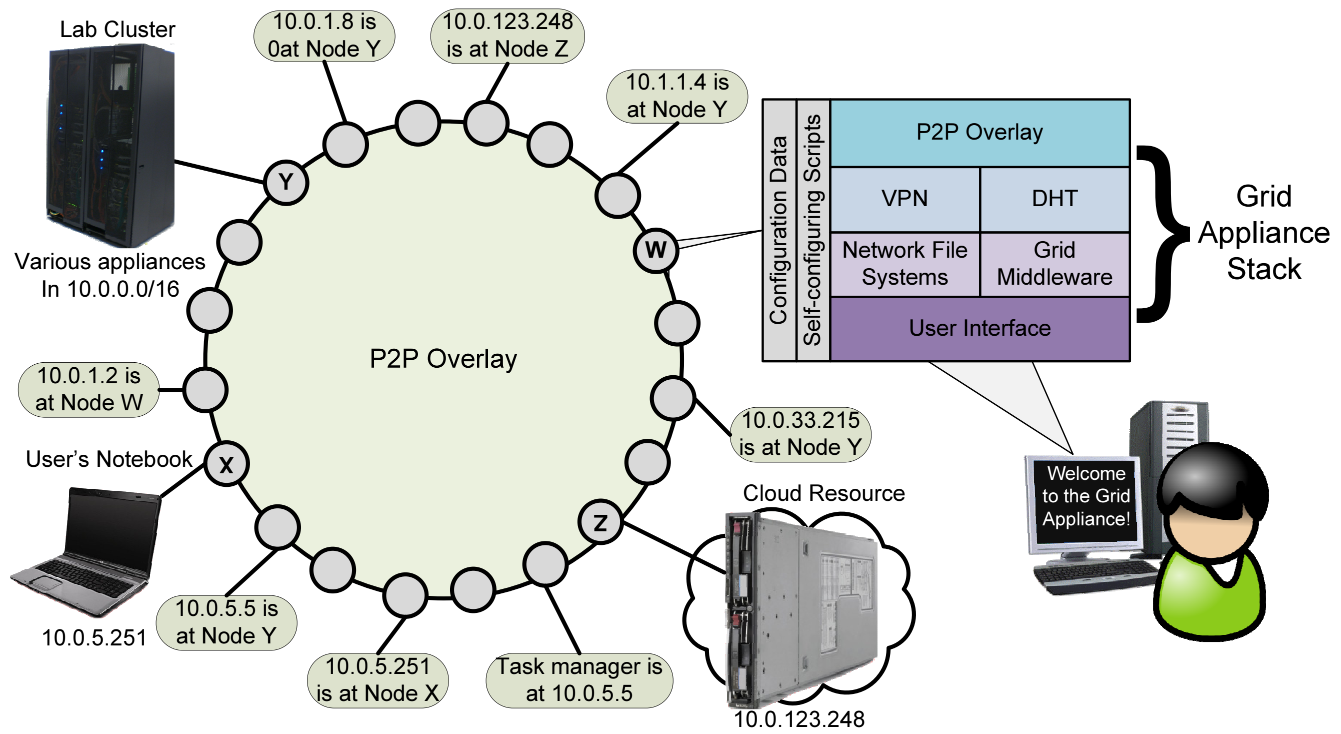

Fig. 1. A "Grid Appliance" system consisting of various users and resource types. The grid uses the P2P overlay for connectivity through a VPN and discovery mechanisms through the DHT. The Grid Appliance software stack consists of P2P software, VPN, DHT, grid middleware with self-configuring scripts, and a user interfaces for the middleware.

virtual private network (VPN), useful for security, supporting unmodified applications across a network overlay, and handling network asymmetries. The DHT is also used for the decentralized coordination of the grid. Users 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 simply starting a virtual machine on the computers intended for use within the grid. 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 a component of the system. User configuration of the system is handled by using social networking like group interfaces, while deployment uses pre-built virtual machine images. A graphical overview of the system is illustrated in Figure 1.

II. Wide Area Overlay Network of Virtual Workstations

This work furthers the vision began by our earlier work wide-area overlay of virtual workstations [5] (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.

A. P2P Overlays

Peer-to-peer or P2P systems create environments where all members of a system are equal. The key feature of most P2P systems is discovery with additional services built on top, such as voice and video with Skype or data sharing with BitTorrent. Many forms of P2P support 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.

Before further discussion, let us introduce our P2P overlay named Brunet [6], 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 [7], commonly used for decentralized BitTorrent, and Dynamo [8], to support Amazon's web site and services.

Brunet support for NAT traversal makes it unique from other structured overlays. Originally in the WOWs [5], Brunet facilitated the dynamic connections amongst peers in the grid. Since then, it has been extended to support DHT with atomic operations [**?**], efficient relays when direct NAT traversal fails, [9], support for network disconnectivity [**?**], and cryptographically secure messaging [9].

B. Virtual Private Networks

A common question with regards to our work is "why do grids need a VPN?" The two primary issues are related to connectivity, the first being the limited address space provied by IPv4, which is quickly approaching its limit. Network address translation (NAT) allows more devices to connect to the Internet but at the cost of symmetry. Two machines on different NAT devices or NATs cannot easily communicate directly with each other without some assistance. 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 all these issues, so long as the user can connect to the VPN, as all traffic is routed through a successfully connected pathway.

The VPN used in the system is called IPOP [9], [10]. IPOP (IP over P2P), as the name implies, builds uses Brunet to route IP messages. By using P2P, maintaining dedicated bootstrap nodes have less overhead, and in fact, our approach with IPOP allows an existing Brunet infrastructure to bootstrap independent Brunet infrastructures in order to isolate IPOP networks in their own environments [**?**].

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 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 int $hash(IP)$.

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 that IP address are directed over that secure link.

In our original design [**?**], the virtual network was secured through a kernel level IPsec stack. A model kept through our first generation Archer deployment. This approach only secures virtual network links between parties and does not secure the P2P layer; furthermore, in IPsec configuration each peer requires a unique rule for every other peer, which limited the maximum number of peers in the VPN. Securing the P2P layer is important, otherwise 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.

For grids that rely upon VPNs to connect resources and users, this can impose the need for a certificate for the VPN and one for the grid. Though in our approach, we avoid this problem by using a VPN that allows a user to verify the identity of a remote peer and obtain its certificate, and have taken advantage of hooks in grid software that are called to verify a remote peers authenticity. 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.

C. Virtual Machines in Grid Computing

Even further back, we were the first to advocate the use of virtual machines (VMs) in grid computing for security and customization [**?**]. In this work and in others since [**?**], [**?**], [**?**], Qualitatively, VMs have been established 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, where processes running inside the VM have no access to the host operating system. Furthermore, VMs can have limited or no networking access as controlled by the host, which effectively seals 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 on the end user being that they have a VM software or manager. 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. VMs are the magic behind cloud infrastructures that provide IaaS or infrastructure-as-a-service, 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 their is overflow, high demands, or the user does not want to maintain their own hardware. 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.

III. The "Grid Appliance" Architecture

Our approach attempts to reuse as many available components to design a grid middleware by doing so, the idea can then be made generic and then applied to other middleware stacks. As a result, our contribution in this paper focuses primarily on a few 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. In this section, we will overview these contributions as we put together the pieces of the "Grid Appliance."

A. Web Interface and the Community

Before deploying any software or configuring any hardware, a grid needs to have some form of organization. Who will be responsible for distributing and signing certificates, who will be able to join the grid, how will user accounts be handled, and how can responsibilities be delegated. These are complex questions, which may not be easy to answer. For less restrictive systems, like a collection of academic labs sharing clusters, it might be very easy. One of the professors would handle the initial authorization of all the other labs and then delegate to them the responsibility of allowing their affiliates, such as students and scholars access.

The greater challenge becomes having the professors or worse yet students actually maintaining the certificates, having certificate requests properly submitted and handled, and placing signed certificates in the correct location. 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, one for direct members of the grid and another for those who have been given access by trusted entities inside the first group, as we will differentiate as the grid group and the user groups throughout the rest of this paper. 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. For example, they might create a 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 will be connected to the grid and able to submit and receive jobs.

Thus to join a grid a user only ever needs to sign onto a web site to download a configuration data, which can then be used on multiple systems. To support this process, the configuration data contains cryptographic information that facilitates the obtaining 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, and if there is, it contacts the web interface with the cryptographic information and a public key. The web interface verifies the user's identity, 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. One of particular interest is not the grid itself but rather a bootstrapping community for grids. 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. Thus when a user comes to our public interface, they have the opportunity to reuse our bootstrap infrastructure and only need to focus on the establishment of their VPN and grid services, which has been trivialized to accepting or denying users access to a group. 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.

B. The Organization of the Grid

The previous section focused on the web interface and how it facilitates the configuration of the grid and skirted the issues of detailed configuration and organization. The configuration of the grid mirrors that of the connection process. The first tier groups map to a common grid and each grid maps to a unique VPN. Thus when a user creates a new grid group, they are actually configuring a new VPN, these details include 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 earlier, 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 queries 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.

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 I, which

	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 [2]	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 I
GRID MIDDLEWARE COMPARISON

covers most of the well established middleware and some recent research projects focused on decentralized organization.

Of the resource management middlewares surveyed, Condor matches closest with our goals due to its decentralized properties and focus on desktop grids. In particular, with Condor, we are able to have multiple submission points, something that would not be trivial to implement in other systems. Additionally, adding resources to Condor can be done without any configuration from the managers. Conversely, in SGE and Torque, after resources have been added into the system, the user must manually configure the manager to control them. Another aspect that makes Condor a reasonable choice is its support for opportunistic cycles. Most scheduling software assumes that resources are dedicated and do not handle cases, where other processes or users interact with the system. Condor, however, can detect the presence of other entities and as a result will suspend, migrate, or terminate a job. All things are not perfect with Condor though, it does require manager nodes, whereas having no manager node would be ideal.

2) Self-Organizing Condor: While the requirement of having a central manager may be concerning, the overhead of running one is small and Condor supports the ability to run many in parallel through the use of "flocking [**?**]." 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. Additionally a manager can easily be setup on a VM, run in the background, and forgotten.

To configure Condor, manager IP addresses are stored into the DHT using the key *managers*. When new peers join, they query the DHT, obtains the list of all managers, and randomly selects one as its primary manager. The rest are set to 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. If no managers are found, the process repeats every 60 seconds. Once a manager has been found, it is checked every 10 minutes to ensure it is online and additional managers that have come online are added to the flock list.

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 step "1," during system boot, 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. For a manager machine, this means registering in the DHT (not shown), while clients

[1]http://www.cs.wisc.edu/condor/CondorWeek2009/condor_presentations/sfiligoi-Condor_WAN_scalability.pdf
[2]http://www.clusterresources.com/docs/211
[3]http://www.sun.com/offers/docs/Extreme_Scalability_SGE.pdf

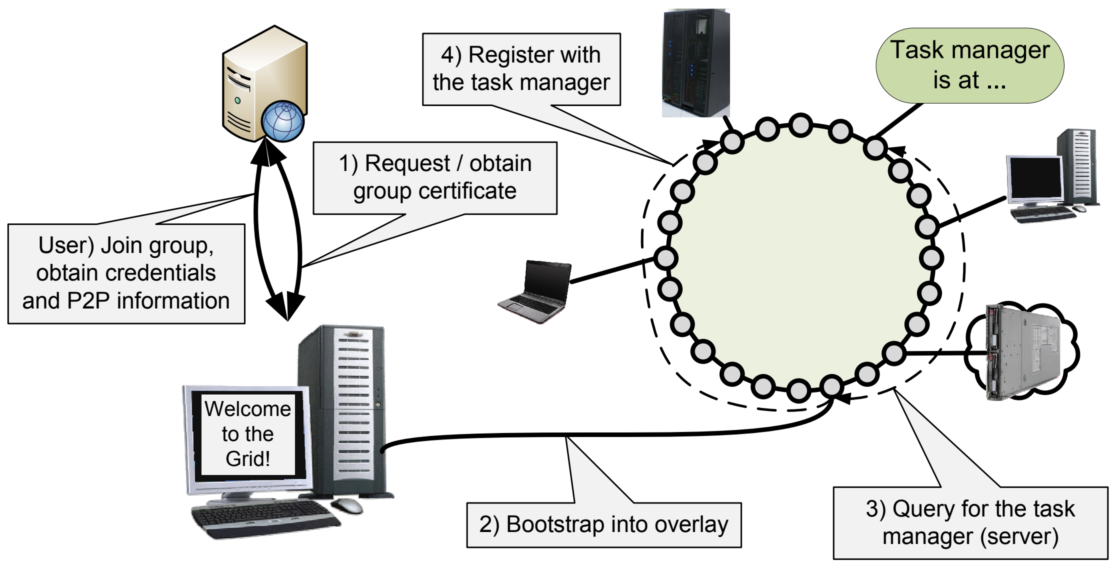
4) Register with the task manager
Task manager is at ...
1) Request / obtain group certificate
User) Join group, obtain credentials and P2P information
Welcome to the Grid!
2) Bootstrap into overlay
3) Query for the task manager (server)

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

and workers search for available managers by querying the DHT, step "3," and then the managers directly, step "4."

C. Sandboxing Resources

As tasks can run on worker and potentially submission nodes, we have devised means to sandbox the environments 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 simulataneously being used for submission. So our sandboxing technique considers more than just locking down the machine but also ensuring a reasonable level of access.

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, network access is limited to the VPN, thus 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 suppor 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 matches that from the website, while the password defaults to "password." 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.

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 lightweight 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) Decentralized Submission of Jobs: From the administrators 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 machines 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, and though this might be clunky as the entire file must be distributed to each worker. This can be an inconvenience if only a small portion of the file is required, which is particularly true with disk images used by computer architecture simulations and applications built with many modules or documentation. To support sparse data transfers, each "Grid Appliance" has a local NFS share exported with read-only permission. There still exists that challenge that traditionally in a Unix system a file systems must be manually mounted. Fortunately, there exist tools to automatically mount file systems, e.g. autofs. The autofs tools work by intercepting file system calls inside a specific directory, parsing the directory link, 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.

IV. A Case Study on 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 and 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.

A. Background

In this sytem, we are configuring two types of grids, a static grid configured by hand and a dynamic grid configured by the "Grid Appliance." The case study follows the creation of a grid originally at the University of Florida that is later extended to Amazon's EC2 and Future Grid at Indian University using Eucalyptus. Each environment has its own type of NAT: 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.

B. Traditional Configuration of a Campus Grid

First, we lay down the infrastructure for connecting the grid, this means choosing a VPN, a requirement since there

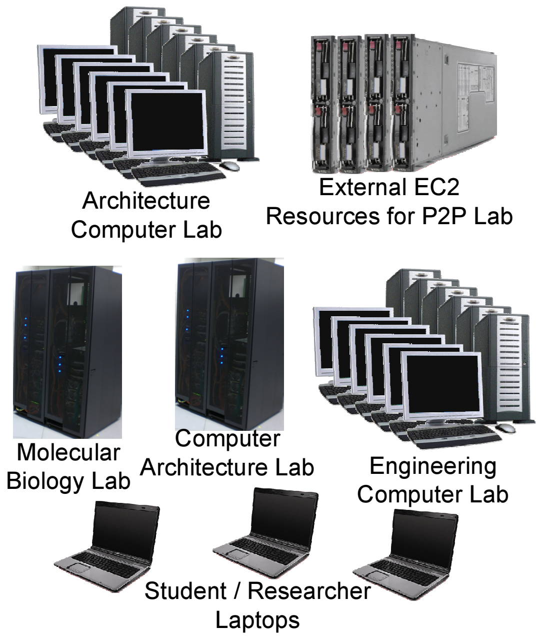
Architecture Computer Lab
External EC2 Resources for P2P Lab
Molecular Biology Lab
Computer Architecture Lab
Engineering Computer Lab
Student / Researcher Laptops

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

are various sets of resources running behind different NATs. There exists a wealth of VPNs available [11], [12], [13] and some explicitly for grids [14], [15], [16]. For simplicity sake, we chose OpenVPN because the other choices require significantly more 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 required, a process 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.

In order to connect to OpenVPN, it must know the servers address and have a signed certificate. 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. The actual dangers in doing so is that if any of the machines were hijacked, the certificate would have to be revoked and all machines would be rendered inoperable. In order to actually create a safe environment for handling certificates, 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.

Finally, we move on to the configuration of Condor, the scheduling tool. The manager has to be allocated first, so that its IP address can be provided to workers and submitters in the system. With regards to submission points, this detail can be left up to the system administrator. 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 having a single submission point creates more work for the system administrator as mentioned earlier. Both approaches have their associated costs and neither is trivial.

Once the details of submitting jobs has been resolved, the next consideration is ensuring homogeneity on the resources. If there are different system configurations on the various machines, an application that works well on one platform could cause a segmentation fault on another, through no fault of the developer, but rather due to library incompatibilities. The easiest way to deal with this approach is to have a common feature set and system layout across all resources. This will require coordination amongst all sites to ensure that the model is followed.

To export this system into various clouds, an administrator starts by running an instance that contains their desires Linux distribution and then installing the grid utilities like Condor and OpenVPN. Supporting individualization of the resources is challenging. The most simple 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.

C. Grid Appliance Configuration of 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, the public one at www.grid-appliance.org or another one hosted on their own resources, similarly the users can deploy their own P2P overlay or use our shared overlay. At which point, users can create their own grids consisting of a shared VPN and different groups for resource priority and delegation.

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. Unlike the other approach the user need not be concerned about the VPNs performance due to routing through the central server or changes in configuration due to the decentralized discovery of managers.

With regards to the homogeneity of the system, 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 that is exported by Condor. Thus a user looking for a specific application, library, or computer architecture can specify that in their job description. Furthermore, 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 V-B, creating "Grid Appliance" from scratch is a trivial procedure.

D. 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. A user could then 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.

E. Quantifying the Experience

The evaluation of these environments focuses on the time taken to dynamically allocate the resources, connect to the grid, and submit a simple job to all resources in the grid. In both systems, a single manager and submission node were instantiated in separate VMs. In the traditional setup, OpenVPN is run from the manager node. Each component in the evaluation was run three times. Between iterations, the submission node and the manager node were restarted to clear any state.

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.

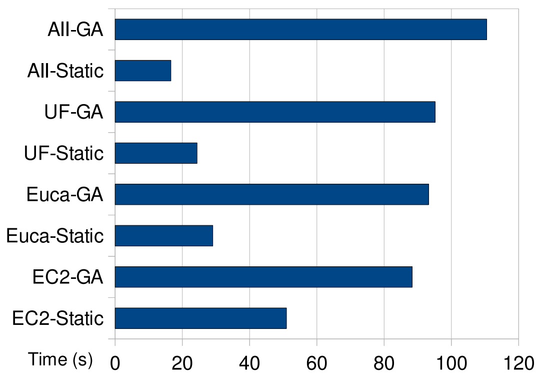

Fig. 4. Comparison of times to construct a grid in various environments using both a statically configured grid and a grid constructed by the "Grid Appliance." Legend: EC2 - Amazon's EC2, Euca - Indiana University's Eucalyptus, UF - Unversity of Florida's ACIS Lab resources, Static - OpenVPN, GA - Grid Appliance.

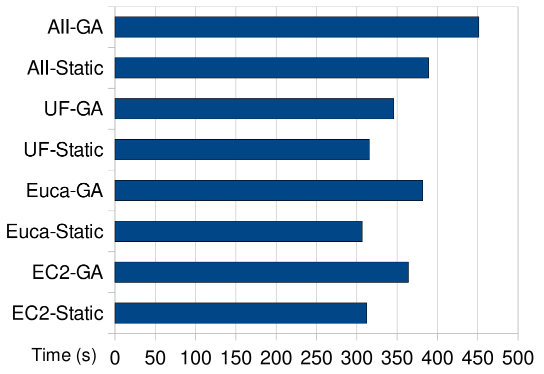

Fig. 5. Comparison of times to run a 300 second job on each resource in various grids configured statically and through the "Grid Appliance." Legend: EC2 - Amazon's EC2, Euca - Indiana University's Eucalyptus, UF - Unversity of Florida's ACIS Lab resources, Static - OpenVPN, GA - Grid Appliance.

In the previous section, we qualified why the approach was easier than configuring a grid by hand, though by doing so we introduce overheads related to configuration and organization. The evaluation verifies that these overheads do not conflict with the utility of our approach. Not only do resources within a cluster install the VMs and connect to the grid quickly, the clouds do as well. While the results were similar, it should be noted that the time required to configure the static approach was not taken into effect. A process that is difficult to measure and is largely reliant on the ability of the administrator and the tools used. Whereas the time for the "Grid Appliance" does include many of these components.

It should be stated that the evaluation only has a single submission node, in a system with multiple submitters, the OpenVPN server could easily become a bandwidth bottleneck in the system as all data must pass through it, which can be avoided using IPOP. Additionally, the current "Grid Appliance" relies on polling with long delays, so as to not have negative effects on the system. Either shrinking those times or moving to an event based system should significantly improve the speed at which connectivity occurs.

V. Lessons Learned

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

A. Deployments

A significant component of our experience stems from the computational grid provided by Archer [17], an active grid deployed for computer architecture research, which has been online for over 3 years. Archer currently spans four seed universities contributing over 500 CPUs as well as contributions and activities from external users. The Archer grid has been accessed by over hundreds of students and researchers submitting jobs totaling over 400,000 hours of job execution in the past two years alone.

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. While Clemson and Purdue are constructing campus grids using the underlying VPN, GroupVPN / IPOP, to connect resources together. 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.

B. Towards Unvirtualized Environments

More advanced users, in particular seed sites of Archer, have expressed desire to run the "Grid Appliance" directly on hardware to avoid the overheads of virtualization as well as the ability to easily deploy resources in the cloud. While creating a shareable cloud instance is possible, physical appliances are not. Due to requests from users wanting better integration with physical machines and as a result of moving away from stackable file and towards creating installable packages. 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 [18].

C. Advantages of the Cloud

We have had the experience of deploying the "Grid Appliance" on three different cloud stacks: Amazon's EC2 [19], Future Grid's Eucalyptus [20], 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 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 simulataneously, we have been able to quickly reproduce problems and isolate them, leading to quicker resolutions, and verification of those fixes.

D. 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 [22]. The appliance at that time ran only on VM consisting of 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 6.

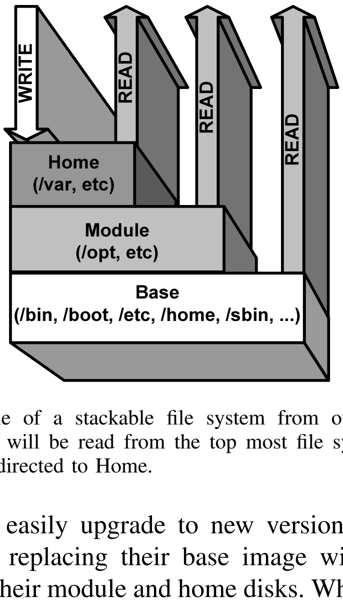

Fig. 6. Example of a stackable file system from our previous "Grid Appliance." A file will be read from the top most file system in the stack and all writes are directed to Home.

Users could easily upgrade to new versions of the "Grid Appliance" by replacing their base image with a new 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 do so, the user would launch into developer mode, where all changes would be written to the module image, rendering the home image unused. 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.

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 [23] to the kernel, adding the weight of maintaining a kernel unto our shoulders. While there does exist FUSE, or filesystem in userspace, solutions, using those properly require modifications to the initial ramdisk, reproduced automatically during the installation of every new kernel, furthermore, our experience with them suggests they are not well suited for production systems.

The approach was also limited to VMs, hosts such as clouds and physical resources cannot, easily, take advantage of it. So while we still think the concept is great, we have removed it from our mainline appliance. Future work for this type of application will be allow users to configure a system and then make it easy for them to create packages. These packages will be significantly more portable and less configuration headaches. Also, to deal with upgrades, we use standard package management, so users do not need to deal with VM disk images to upgrade.

E. Guaranteeing Priority in Owned Resources

In Archer, it was very obvious that members of seed universities should have priority on the resources at their university. Similarly, users should have priority on the resources that they contribute. To support user and group based priorities, Condor has mechanisms that can be enforced at the server that allow for abitrary 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.

F. Timing in Virtual Machines

Certain applications, particularly license servers, are sensitive to time. When constructing a grid consisting of many distributed resources, there is a good possibility that time will be wrong somewhere. With regards to VMs, VMWare [24] suggests synchronizing with the hosts time and to avoid using services like NTP (network time protocol), which may have adverse affects on timing inside the virtual machine. Our experiences recommend the opposite, we have experienced drastic timing jumps and significantly off virtual clocks due to host clocks being set incorrectly. It seems the virtual clock would force an immediate jump in timing for the virtual clock, which could be quite significant if the VM had not been scheduled recently enough. NTP, on the other hand, gradually corrects time. This major jump would cause the system to completely stall due to software that was unable to handle the unreliability in time. Because in many cases, the VMs run on machines that are not owned by the VM owner, correcting the host time is not possible. Our conclusion is that developers should be wary when using virtual timing, especially if they are constructing distributed systems. Rather they should consider using NTP, especially if your application will run on a device where the user may not have the ability to change the hosts timing. It should be noted that the paper from VMWare does not explicitly recommend against NTP, just that it can be difficult to properly to configure. For example, if a solitary NTP server is chosen, it may be offline or behave erratically, an issue typically bypassed when using the default NTP server provided the operating system distributor.

G. 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. [25] 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.

VI. Related Work

Existing work that falls under the general area of desktop grids/opportunistic computing include Boinc [26], BonjourGrid [27], and PVC [25]. 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 are ensuring that submission sites can exist any where not being confined to complex configuration or highly available, centralized locations; ability to dynamically add and remove resources by starting and stopping an appliance; and the ability for individual sites to share a common server or to have one or more per site 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 in organization to OurGrid [28], though whereas OurGrid requires manual configuration amongst sites and networking considerations to ensure communication amongst sites, the "Grid Appliance" transparently handles configuration and organization issues with a web interface and P2P overlay and a VPN to handle network constraints.

In the space of clouds, there exists contextualization [29]. Users are 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 contacts a third-party contextualization broker to receive this file. This approach has been leveraged to create dynamic grids inside the Nimbus cloud [30]. This approach can reproduce similar features of the "Grid Appliance" such as automated signing of certificates and self-configuration scripts in the "Grid Appliance." Though this approach is only sufficient for isolated, single user systems as it does not consider the issues of collaboration amongst disparate groups and connecting resources across clouds and local resources.

VII. Conclusions

In this paper, we have described a novel grid architecture that enables both wide area and educational grid middleware. Our approach focuses on reducing the entry barrier to constructing wide-area grids, rather than just providing a grid, i.e., teaching users how to create grids rather than providing access. The features of the "Grid Appliance" significantly reduce the work that traditional methods of constructing grids would take. We presented this qualitatively and quantitatively in Section IV: 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 conveniently 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 ot other systems. For example, more complex approaches to grids invovle entities known as virtual organizations. A virtual organization allows the same set of resouorces 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.

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] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for high throughput computing," *SPEEDUP Journal*, June 1997.

[2] Cluster Resources. (2010, July) Torque resource manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php.

[3] Sun. (2010, July) gridengine. http://gridengine.sunsource.net/.

[4] Globus Alliance. (2010, July) Globus toolkit. http://www.globus.org/toolkit/.

[5] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "Wow: Self-organizing wide area overlay networks of virtual workstations," in *(HPDC)*, June 2006.

[6] P. O. Boykin and et al., "A symphony conducted by brunet," http://arxiv.org/abs/0709.4048, 2007.

[7] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *IPTPS '02*, 2002.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. New York, NY, USA: ACM, 2007, pp. 205–220.

[9] D. I. Wolinsky, K. Lee, P. O. Boykin, and R. Figueiredo, "On the design of autonomic, decentralized vpns," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2010.

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

[11] LogMeIn. (2009) Hamachi. https://secure.logmein.com/products/hamachi2/.

[12] J. Yonan. (2009) OpenVPN. http://openvpn.net/.

[13] G. Sliepen. (2009, September) tinc. http://www.tinc-vpn.org/.

[14] X. Jiang and D. Xu, "Violin: Virtual internetworking on overlay," in *Intl. Symp. on Parallel and Distributed Processing and Applications*, 2003.

[15] M. Tsugawa and J. Fortes, "A virtual network (vine) architecture for grid computing," *International Parallel and Distributed Processing Symposium*, 2006.

[16] A. I. Sundararaj and P. A. Dinda, "Towards virtual networks for virtual machine grid computing," in *Conference on Virtual Machine Research And Technology Symposium*, 2004.

[17] R. J. Figueiredo, P. O. Boykin, J. A. B. Fortes, T. Li, J. Peir, D. Wolinsky, L. K. John, D. R. Kaeli, D. J. Lilja, S. A. McKee, G. Memik, A. Roy, and G. S. Tyson, "Archer: A community distributed computing infrastructure for computer architecture research and education," in *Collaborative Computing: Networking, Applications and Worksharing*, vol. 10. Springer Berlin Heidelberg, 2009, pp. 70–84. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03354-4_7

[18] D. I. Wolinsky, Y. Liu, P. S. Juste, G. Venkatasubramanian, and R. Figueiredo, "On the design of scalable, self-configuring virtual networks," in *IEEE/ACM Supercomputing 2009*, November 2009.

[19] (2009) Amazon elastic compute cloud. http://aws.amazon.com/ec2.

[20] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009.

[21] K. Keahey and T. Freeman, "Science clouds: Early experiences in cloud computing for scientific applications," in *Cloud Computing and Its Applications*, 2008.

[22] D. I. Wolinsky and et al., "On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations," in *VTDC*, 2006.

[23] C. P. Wright and E. Zadok, "Unionfs: Bringing file systems together," *Linux Journal*, no. 128, pp. 24–29, December 2004.

[24] VMware, Inc., "Timekeeping in vmware virtual machines," http://www.vmware.com/pdf/vmware_timekeeping.pdf, 2008.

[25] A. Rezmerita, T. Morlier, V. Neri, and F. Cappello, "Private virtual cluster: Infrastructure and protocol for instant grids," in *Euro-Par*, 2006.

[26] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *the International Workshop on Grid Computing*, 2004.

[27] H. Abbes, C. Cérin, and M. Jemni, "Bonjourgrid: Orchestration of multi-instances of grid middlewares on institutional desktop grids," in *IPDPS*, 2009.

[28] N. Andrade, L. Costa, G. Germglio, and W. Cirne, "Peer-to-peer grid computing with the ourgrid community," in *in 23rd Brazilian Symposium on Computer Networks (SBRC 2005) - 4th Special Tools Session*, 2005.

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

[30] A. Harutyunyan, P. Buncic, T. Freeman, and K. Keahey, "Dynamic virtual AliEn grid sites on nimbus with CernVM," *Journal of Physics: Conference Series*, 2010.