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 OpenScience Grid provide researchers vast amounts of compute resources but with limited access, potentially long job queues, and environments and policies that might require changes to a user's work flows. In many scenarios, individual users and communities can benefit from less restrictive more ad-hoc grids with flexibility in deploying resources, sharing access, and configuring the environment, even if this results in a smaller scale grid.

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 configuration, decentralized user access, and web interfaces to construct grids. This paper presents the "Grid Appliance," originally presented as a tool 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. This paper presents the architecture of the system, lessons learned during the development of WOW to "Grid Appliance," and a case study backed by quantitative analysis that verifies the utility of our approach.

I. Introduction

Grid computing presents opportunities to combine various distributed resources together to form powerful computing systems. Due to the challenges in coordinating the organization of grids, researchers typically become members of existing grids. However, often times existing grids are limited in terms of flexibility and policies provided, and users resort to creating and managing their own local systems. While there is a wealth of middleware available, including resource managers like Condor [1], Torque (PBS) [2], and Sun Grid Engine [3], most researchers see the entry barrier to installing and future management of these systems as being greater than their usefulness and as a result often times turn to inefficient ad-hoc resource discovery and allocation.

After assembling a local system, users working in the same field or in different departments or groups within the same organization may desire to combine their individual resources into even larger grids. Some of the resource managers listed early support simple grids, while other individuals turn to more complete solutions like the Globus Toolkit [4] and 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 "Cloud" computing by means of infrastructure-as-a-service, small computer clusters can grow dynamically into much larger systems. Doing so has demands similar to that of creating a distributed grid in particular with emphasis on connectivity and organization. This is because the creation of an individual cloud instance for grid computing will typically bind it to a specific grid, while the support for dynamic grids may be too imposing.

In this paper, we focus on these concerns and present means to support a simplified model for the creation of ad-hoc, distributed grids. We have implemented a system supporting these ideas in the "Grid Appliances," which as will be demonstrated in this paper, allows users to focus on making use of the grids while minimizing their efforts in setting up and managign 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.

Our 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. In this paper, we extend those ideas to establish means for the dynamic creation of a grid with support for security, decentralized access, and user-friendly approaches to grid management have many independent groups that do not share all resources with members of the grid . In addition, we present several years worth of experience in designing and developing our reference implementation, the "Grid Appliance."

Our experience derives from the development of the "Grid Appliance" and its use in several projects. Of these projects, Archer, a distributed grid for computer architecture research spanning 6 seed universities, stands out. Archer allows any academic access to contribute and use resources with little time overhead. The users join a website, download a configuration image and a virtual machine (VM), and start the VM. As soon as the VM has completed the booting process, the user is connected to a set of distributed resources and able to begin submitting and receiving 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

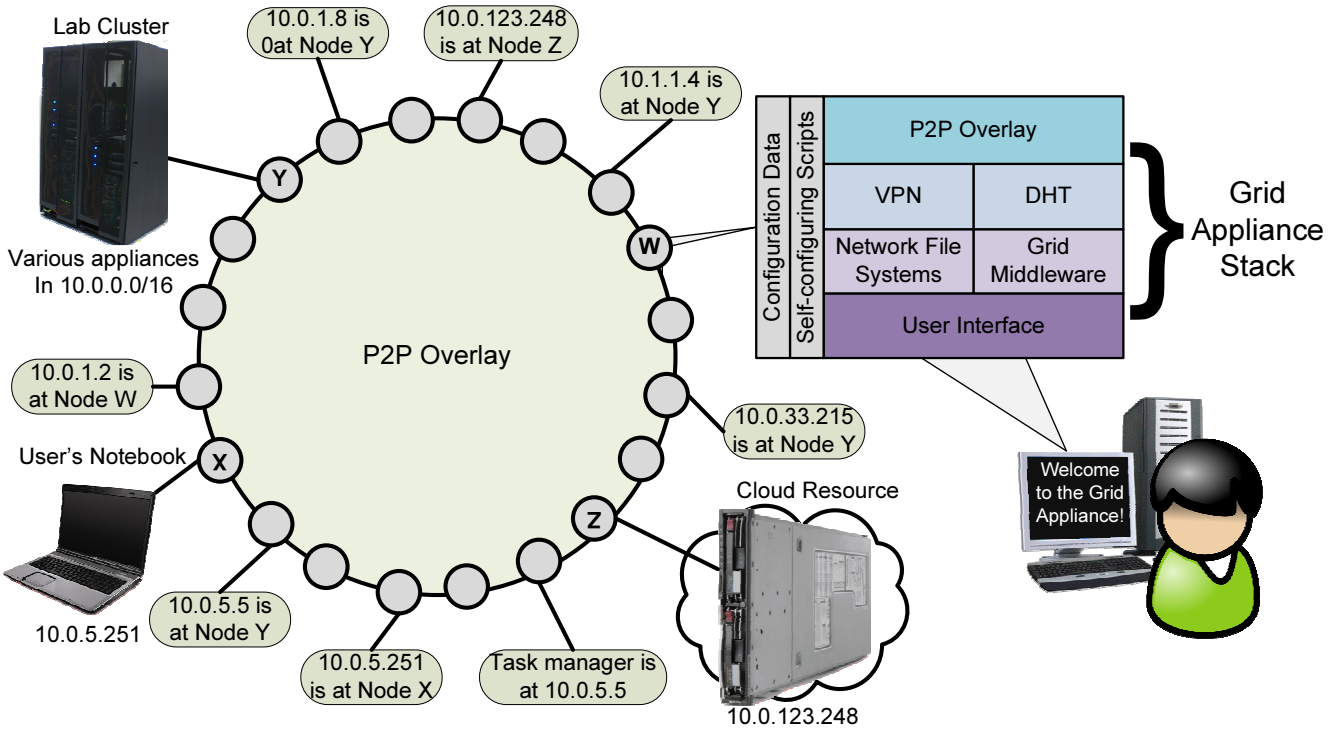

Fig. 1. A "Grid Appliance" system consisting of various users and resource types. The grid uses the P2P overlay for connectivity through a VPN and discovery mechanisms through the DHT. The Grid Appliance software stack consists of P2P software, VPN, DHT, grid middleware with self-configuring scripts,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

original purpose of the DHT was to support a decentralized 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. At the basis, it 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 interaction with the system focused on two aspects configuration of the grid using social networking like group interfaces and deployment of virtual machine images. A graphical overview of the system is illustrated in Figure 1.

To justify our methods, consider the difficulty in combining resources across disparate networks, which may or may not involve multiple research groups. The set up of security, connectivity, and the continuous management of the system may require an information technology (IT) expert. Network constraints present another complexity beyond configuration and organization of distributed resources. Contributing groups may have resources behind different network address translation devices (NATs) and firewalls, preventing direct communication with each other. Even assuming that an institution's network administrator is willing to make exceptions for the grid, additional rules may be required for each new cluster or resource added internally and externally, quickly becoming unmanageable. Our approach embraces decentralized systems behind NATs and novice grid administrators.

The rest of the paper is as follows. Section **??** overviews the core components of a WOW. In Section III, we describe the key features of our approach to creating WOWs that distinguishes them from previous work. Section VII provides a case study of a grid deployment using standard grid deployment techniques compared to our "Grid Appliance," describing qualitatively the benefits of this approach. Using the system described in the previous section, Section VIII presents a quantitative evaluation of the time taken to self-configure a "Grid Appliance" wide-area virtual clusters. We share our experiences from this long running project in Section IX. Finally, Section X compares and contrasts other solutions to these problems.

II. Wide Area Overlay Network of Virtual Workstations

This section presents an overview of our previous work and the core technologies used by the "Grid Appliance." These include the P2P overlay software and the VPN and the use of virtual machines in grid computing. A WOW [5] combines virtual machines and virtual networking to support dynamic sizes of a grid, which recently has been furthered by Abrahams et al. [**?**] to verify support for extending into the cloud. This section covers the development of WOWs over the years as it relates to our other publications and as means to distinguish our contributions 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 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.

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 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, how can responsibility be delegated. These are complex questions, which might not necessarily have an easy 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 actually maintaining the certificates, having certificate requests properly submitted and handles, and having signed certificates placed in the right 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 heirarchial, our model consists of a two layers, one for access to the grid and another to differentiate the various subgroups or delegated responsibility in the group. Once a user has access to a subgroup, they have the ability to download the grid configuration data that can be added to a "Grid Appliance." As will be described later, this data is used to obtain their credentials for machines that they use and connect them to the grid.

To give an example, consider our computer architecture grid, Archer. Archer was seeded initially by the University of Florida, so we became the creators of the initial Archer grid and are maintainers of that group. As new universities and independent researchers have joined Archer, they request access to the "Archer Grid" group. Upon receiving approval, they then need to form their own subgroup so that they so that they can allow others to connect to the grid. To do so, they will create a second layer group called "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 need 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 enables a user to obtain a signed certificate from the web interface through XML-RPC over HTTPS. This process is handled either by booting the "Grid Appliance" or restarting a "Grid Appliance" service. 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 community of sorts. Not necessarily a grid community but a boot strapping community for grids. A single web interface can easily play host to many individual grids but the same underlying P2P infrastructure can easily be shared amongst all of them due to the unique nature of Brunet to support recursive bootstrapping. Thus when a user comes to our publicly shared interface, they need to only 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 actually are 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 configure subgroups within the grid and delegate responsibility. It is also the location that users download their configuration data. Users can download either download a configuration data for a submission, worker, or manager node. 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.

Our grid composition is largely based upon a desire to support a decentralized environment, while still retaining reliability and a strong user base. 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 covers most of the well established middleware and some recent research projects focused on decentralized organization.

Of the resource management middlewares surveyed, Condor matched closest with our goals due to its decentralized prop-

[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

	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

erties 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 processors or a user also interact with the system. Condor, however, can detect the presence of other processes or a user and 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.

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.

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.

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.

IV. User Experience

The previous section covered aspects of the architecture important to administrators. This section concludes the core architecture by focusing on those features important to users. This includes components to make the environment more desktop grid friendly and better support for decentralized task submission.

A. Respecting Host Usage in a Desktop Grid

In desktop computing grids, a grid resource may duplicate as a workstation directly used by individuals. If the grid software runs in a VM, for sand boxing purposes, it will be unable to detect an active user on the host. 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.

B. Decentralized Submission of Jobs

While grid middlewares typically provide data distribution mechanisms, these can be clunky, requiring that the entire file 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. 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.

V. The "Grid Appliance": From a User's Standpoint

This section highlights the different aspects of the system as shown in Figure 2. This presents what a first time user would experience including interaction with the website and the services used directly or indirectly to configure a working grid system. Later on in Section VII, this effort will be compared to the configuration and use of a manually configured grid.

A. Creating the Grid

Prior to deploying a grid, users should be aware of the core user components of the "Grid Appliance": understand the group website, able to deploy VMs or physical resources, and able to use a grid task scheduler (discussed in Section **??**). To address users who may not be familiar with these parts, helpful tutorials are provided on www.grid-appliance.org.

The process begins with with the step labeled "User." Users can either create a new grid or join an existing one via groups. There are two types of groups: "GroupVPN" groups and "GroupAppliance" groups. A "GroupVPN" group constitutes a virtual private network that provides connectivity within a grid and isolation from other grids, while "GroupAppliance" groups represent subsets of a grid. This approach allows for delegation of responsibilities across the grid and as a means to ensure higher priority for members of the same "GroupAppliance." The latter is important in a scenario where multiple independent departments at a university or group collaborators across multiple institutions may desire to share resources together, to take advantage of each others idle cycles. Though when a deadline approaches, they will want priority on their contributed resources. Without priority, they may avoid a grid altogether. Though with the "Grid Appliance," the "GroupVPN" allows the departments to collaborate their resources into a common university grid, while still maintaining priority to their own resources through "GroupAppliances."

Creating a new grid involves the creation of a new a "GroupVPN" group followed by a "GroupAppliance" group. Joining an existing grid requires a user to join its "GroupVPN" group and then a "GroupAppliance" group matching their affiliation or creating their own. The "GroupAppliance" group also provides configuration files to create managers, workers, and clients. The configuration file is key to enabling automation of the appliance. The contents assist the "Grid Appliance" scripts in taking an unconfigured appliance to a configured one through discovery and coordination using the DHT, as

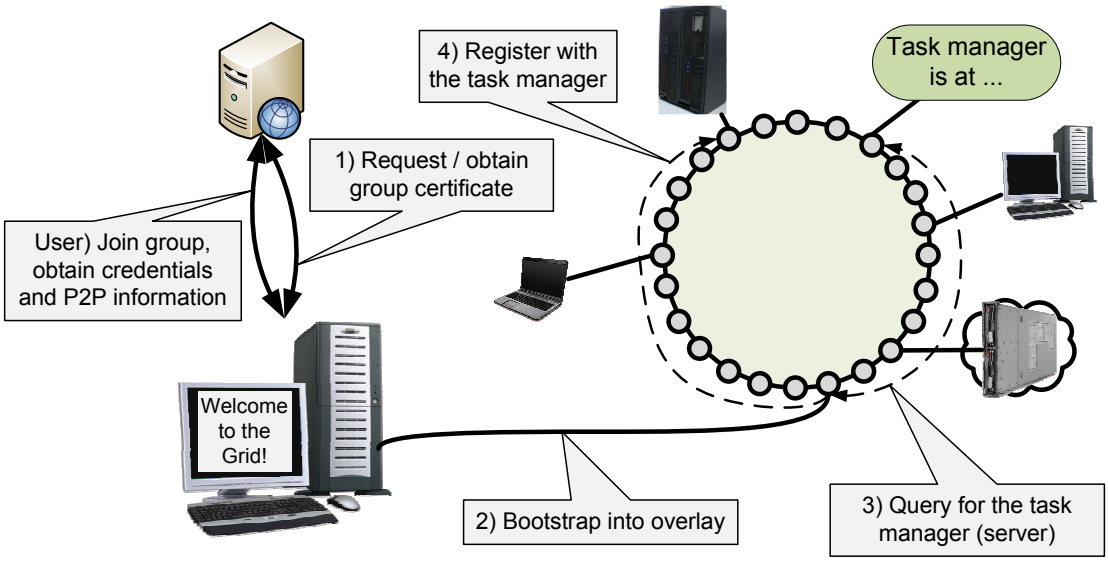

Fig. 2. An example deployment scenario: obtaining configuration files, starting the appliance, and connecting with a resource manager.

described in Section **??**. The contents of the configuration files are as follows:

P2P Config:
- List of bootstrap nodes
- P2P security configuration (key, certificate)

VPN Config:
- VPN Group name
- Group Web interface URL
- Secret key for obtaining signed certificates
- Group user name

Grid Appliance config:
- Appliance type (manager, work, client)
- Administrator ssh key

A new grid requires a manager to coordinate workers and clients. In the context of traditional cluster and grid middleware, this is the machine that keeps track of the global task queue and enforces user priorities. Machines used only to run tasks are called workers. Finally, client systems have the ability to submit tasks and other features useful for interacting with the grid in addition to the roles of a worker. While these terms are used loosely in this discussion, their use is determined by the middleware used. Section **??** presents their relationship when used with Condor.

With configuration files in hand, users are ready to deploy a "Grid Appliance" grid on physical, virtual, or cloud machine. Users can either download a preconfigured VM, start a cloud instance, or create their own system through the use of package management systems, like APT and YUM. To create a new system, users can take an existing Linux installation, add the "Grid Appliance" repository, and install the packages by executing commands such as "yum install grid-appliance" or "apt-get install grid-appliance." Alternatively, this process can be automated during the installation of the operating system through a preseed or kickstart file. Grid configuration files can be added to the system via a floppy disk for a virtual or physical machine, user data for cloud instances, or placed in a specific directory inside the system or on a local website for all three types of configurations. This too can be streamlined with operating system installation or done afterward.

The requirements for deploying one's own grid are to instantiate a manger, some workers, and at least one client. From Figure 2 step "1," during system boot, without user interaction, each machine contacts the group website to obtain a valid "GroupVPN" certificate. This approach allows a single floppy disk to bootstrap many systems. Once the machine has a certificate, 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 hides from the user the complexity of setting up a common fabric that is subject to network dynamics and connectivity, issues further explored in Section VI. 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 and workers search for available managers by querying the DHT, step "3," and then the managers directly, step "4." The technical details for our approach with Condor are described in Section **??**.

Considering these three issues: submission site, job queue site, and API requirements, we firmly believe that out of the potential choices Condor matches best. While systems like PastryGrid and BonjourGrid are developing nicely, our attempts to get PastryGrid online failed as Pastry suffers from dynamic bootstrapping issues and PastryGrid was unable to actually execute our submitted tasks and both do not support priority nor fairness. Also, systems like PBS/Torque and SGE are significantly centralized. While using systems that allow cross-domain grids through middleware like Globus [4] could potentially enable them to meet our goals, we believe the end result is too complex. Boinc is inappropriate for our approach

as it works best when used to facilitate a single project from a central point.

VI. The Motivation for VPNs

Using the aforementioned techniques "Grid Appliances" can be constructed in one of two ways: local and wide area. The "Grid Appliance" ships with two default configurations, one that connects users to a globally available public system and another that allows for LAN only grids. Local grids can be constructed by booting the appliances, which will then use multicast self-discovery to find other resources, create the DHT overlay, and then form VPN links. Alternatively, the user can connect to the default publc system or use "GroupAppliances" to create and manage their own grid, both of which bootstrap from a public shared DHT overlay. This does not prohibit more advanced users from downloading our "GroupAppliances," as its available as a VM, and host their own DHT overlay.

VII. A Case Study on Deploying a Campus Grid

We now present a case study to explore the qualitative differences in deploying a campus grid 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. In this case study, we examine approaches to setting up a grid connecting these different sets of resources using commodity components in contrast to the "Grid Appliance."

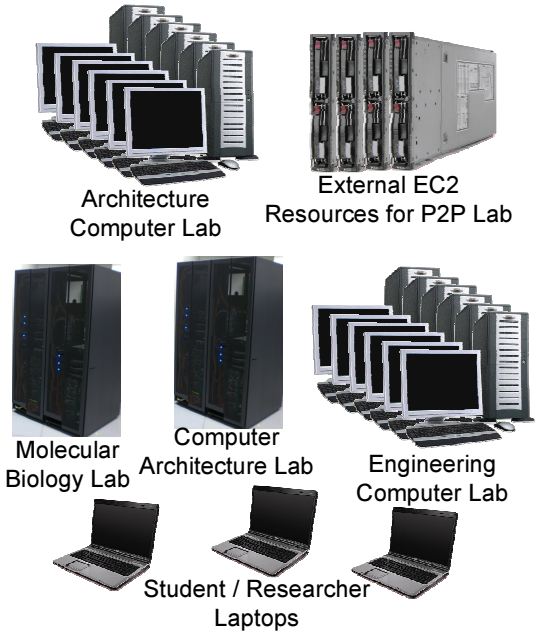

Fig. 3. A collection of various computing resources at a typical university.

A. Traditional Configuration of a Campus Grid

The first step in constructing the grid is determining the network configuration of the system. Condor like most other push based schedulers requires that the submission sites have direct continuous network access to the workers in the system. To deal with potential network asymmetries, all systems can be placed on a common VLAN, though this may be complicated across a campus and even more difficult if resources were to include other universities or cloud resources. To deal with these potential network asymmetries, the user could need to deploy a VPN. The approach requiring the least overhead would be to deploy an OpenVPN on the manager node. Using OpenVPN, like most VPNs, requires that each node be pre-configured with a unique key and signed-certificate, a tedious process that takes time and effort. The problem with using most VPNs is there lack of support for dynamics in the system. If a node in the critical path between two nodes goes offline, network communication will be broken. The only VPN resilient to these types of failures is IPOP as there are no critical nodes between two communicating peers.

The next issue is job submission. A global submission site allows for a simpler though not necessarily trivial network configuration. Having each user supply their own submission site requires all-to-all connectivity in the system, which is not very common on a university campus. So while a global submission site may reduce networking configuration costs, it increases administration costs both in terms of utilities and personnel. Although a global submission site requires administration, Boinc makes claims that a fully functional system should require approximately 20 hours a week of effort to maintain this system. Using a global submission site will also cost computing and electrical resources, as more users are added to the system, file system space, memory, and processing power may be needed on the submission site. Alternatively, if each user supplies their own submission site, all these issues will be negated though all the users will need to maintain uninterrupted connectivity to the worker machines. Regardless of the path taken, someone will have to sign a new certificate for each resource a user wants to connect to the resource or create a new account for each new user.

Once the details have 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.

Each resource in the grid needs to be configured with the address of the global job queue or manager. In Condor, this can be multiple individual queues as well as be separate from individual submission sites. Condor supports a high availability mode as well as having parallel queues via flocking. Regardless of which approach is taken, the IP address or hostname of these end points will need to be added to each of the resources. If they ever change, each machine will need to be reconfigured to point to the new locations.

The final component deals with fairness, which has two aspects: users who contribute resources should have priority on them and if a user is directly accessing a computer lab's resource directly it should not be hampered by remote users' jobs. To support user and group based priorities, Condor has mechanisms that can be enforced at the server that allows for abitrary means to allow one user to have higher priority than another user on a specific machine. This is done through specifying on the worker machine two variables, the contributing users name and/or the contributing groups name. When the server receives a job request, it compares the submission nodes user name / group name with that of workers, if there is a mapping that user will get priority over other users. This can either be realized by the next time the resource is available, it will be given to that user or by pre-empting a non-owners job from that resource.

The format for this configuration is as follows:

Job queue (server):
  NEGOTIATOR_PRE_JOB_RANK = 10 * (MY.RANK)
Worker:
  GROUP_RANK = TARGET.Group =?= MY.Group
  USER_RANK = TARGET.User =?= My.User
  RANK = GROUP_RANK || USER_RANK
Worker and Submitter:
  Group = "Group's Name"
  User = "User's Name"

Even with this in place, there needs to be a mechanism to verify the identity of a user, so that a malicious user can not feign another identity and obtain unfair priorities. In the "Grid Appliance," this is addressed by storing this information inside the certificate of the VPN, which the server can request using a XMLRPC link to the P2P VPN. In other systems, this may require accessing a centralized user database or requiring all users in Condor to use a certificate.

Finally, resources running on a campus computing lab must be multiplexed with a user accessing the machine directly, such that grid resources do not overwhelm the local user. If the resource use a VM, then an administrator will need to manually configure means to ensure that tasks in the VM do not inhibit local use of the system.

B. Grid Appliance Configuration of a Campus Grid

All these considerations 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 VPN groups for different grids and then their grid groups to ensure priority on their own resources.

The website enforces unique names for both the users and the groups. Once the user has membership in a "GroupAppliance" group, they can download a file that can be used automatically configure their resource. This means obtaining a signed certificate, configuring the group information in Condor, connecting to a decentralized VPN, and discovering the server in the grid. The user does not need to be concerned about location thanks to the VPN or changes in the configuration of the grid thanks to decentralized discovery of the server. Finally, the "Grid Appliance" approach ensures homogeneity as all users install the same packages on the same platforms. Whereas a traditional grid would require users to conform with each other or deal with the incompatibilities across machines.

C. Comparing the User Experience

After a user has obtained an account and done all the other appropriate steps to connect with the grid, in the traditional setup, they will SSH into the submission site. Their connectivity to the system is instantaneous, their jobs will begin executing as soon as it is their turn in the queue, which can be instantaneous in a lowly utilized system.

The procedure taken by the "Grid Appliance" is slightly different. When the user first boots a "Grid Appliance," sometimes it will have already finish the connection procedure prior to the user accessing a command prompt and sometimes not. Typically a "Grid Appliance" will be completely ready within 30 seconds or less, though our current approach relies on polling the state of the P2P overlay and the VPN rather than using events, which may further lower this time. To ensure users that everything is progressing normally, we have a window presenting the state of the system, in particular the VPN and Condor.

Once a user has access to a prompt, they can submit jobs. Their jobs will too begin executing as soon as it is their turn in the queue; however, before a job can be executed, a direct VPN link must be established between the submission site and the task worker. The amount of time required varies on the network configuration, though in all cases a direct link will be established. Sometimes that direct link consist of routing through a well chosen proxy as discussed in [9].

With the "Grid Appliance," users are not limited to accessing their files through SSH and SFTP. We have also configured the "Grid Appliance" to support a local Samba mount or Windows file share. Something recognized as not being safe to do on an open / untrusted network but is safe to do since the "Grid Appliance" runs on the users local resources.

VIII. Evaluation

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. This section verifies that these overheads do not conflict with the utility of our approach. This evaluation's goal is to determine the time required to start a grid and run a task on the connected resources individually at multiple sites individually and then cumulatively. We compare a statically configured grid versus our dynamic "Grid Appliance." This evaluation does not include the time required to prepare a static environment, including the signing of VPN certificates nor specifying the head node at each individual resource, as these tasks are

not automatable and require individual configuration by the grid deployer. Additionally, 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 GroupVPN.

The three environments chosen are Amazon's EC2 supporting a simple 1:1 NAT, University of Florida directly behind an "iptables" NAT and then a Cisco NAT, and finally Future Grid at Indiana University using Eucalyptus behind what appeared to be an "iptables" NAT. Prior to beginning the evaluation a manager and submission node were started and connectivity between the two was verified. In the static system, OpenVPN is run from the manager node. The evaluations were run three times. Between iterations, the submission node and the manager node were restarted to clear any state. We measured 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 latter 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 GroupVPN's dynamic link creation 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.

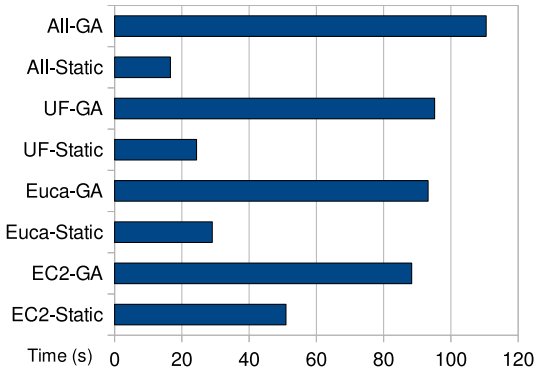

Fig. 4. Comparison of times to construct a grid in various environments using both a statically configured grid and a grid constructed by the "Grid Appliance." Legend: EC2 - Amazon's EC2, Euca - Indiana University's Eucalyptus, UF - Unversity of Florida's ACIS Lab resources, Static - OpenVPN, GA - Grid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

The results make it very clear that the overheads introduced through a decentralized, P2P VPN are minimal and that the time to connected is not significantly longer than a statically configured environment with a centralized VPN. 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.

IX. Lessons Learned

This section highlights some of the more interesting developments and experiences we have had that did not fit in

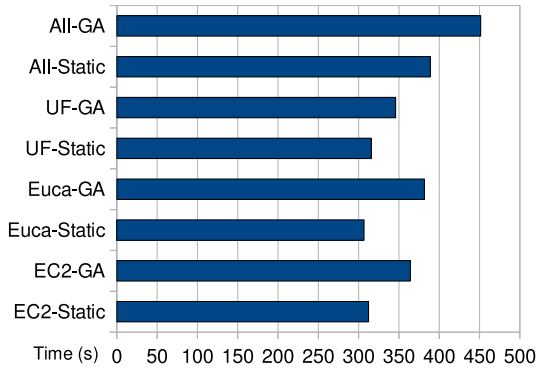

Fig. 5. Comparison of times to run a 300 second job on each resource in various grids configured statically and through the "Grid Appliance." Legend: EC2 - Amazon's EC2, Euca - Indiana University's Eucalyptus, UF - Unversity of Florida's ACIS Lab resources, Static - OpenVPN, GA - Grid Appliance.

the topics discussed so far. A significant component of our experience stems from the computational grid provided by Archer [11], 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. Recently, several private small-scale systems have come online 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.

A. 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 [12]. 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.

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

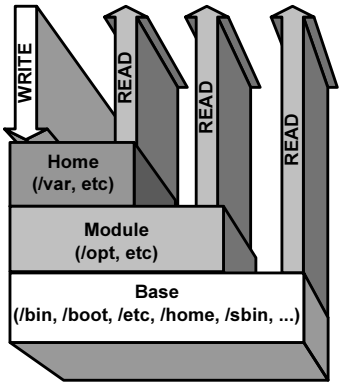

Fig. 6. Example of a stackable file system from our previous "Grid Appliance." A file will be read from the top most file system in the stack and all writes are directed to Home.

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 [13] 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.

B. 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 [14] 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.

C. 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, because there would be two network interfaces connected to the same address space but not the same network. One potential solution is for a user to run the VPN behind a NAT, like inside a VM behind a VM NAT or a cluster behind a NAT device.

Users of the "Grid Appliance" should not have to concern themselves with this issues. Prior work on the topic by Ala Rezmerita et al. [15] recommends using the experimental address class E ranging between 240.0.0.0 - 255.255.255.254, though this requires Linux kernel modifications. Though 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 also limited deployment in physical and cloud machines. 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.

D. Securing the VPN and Overlay

Our original design secured the virtual network through a kernel level IPsec stack. A model kept through our first generation Archer deployment and GroupVPN. The problem with this approach is that securing links of the VPN only partially secures the P2P VPN and using IPsec, which is not

trivially configured, to pre-configured environments. 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.

To address these, we implemented a security system directly into the P2P system, thus securing both VPN and P2P communication. The security filter supports both DTLS and a protocol similar to IPSec. The overheads of using security turned out to be significantly small, as we explored in [9].

E. Towards Unvirtualized Environments

Users comfortable with the system and wanting to dedicate computers to their "Grid Appliance" grid wanted to remove the overheads of using a VM as well as being able extend their compute resources into the cloud. So while, we could create shareable cloud instances, but a physical machine is a little more complicated as it requires specialized non-standard software to create a shareable image. Especially since other machines may have different hardware configurations that may actually prevent reusing a shared image.

Due to requests from users wanting better integration with physical machines and as a result of moving away from stackable file, we move towards creating installable packages. Also, we improved the VPN to support a router mode, whereby a single VPN can be used by many machines inside a LAN. The implications of packages mean that users can easily produce "Grid Appliances" from installed systems or during system installation. While the VPN router allows resources inside a LAN to 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, this was discussed more in our previous work [16].

F. Advantages and Challenges of the Cloud

We have had the experience of deploying the "Grid Appliance" on three different cloud stacks: Amazon's EC2 [17], Future Grid's Eucalyptus [18], and Future Grid's Nimbus [19]. 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.

Beyond the use of extending into clouds for on-demand resources, they are also very convenient for debugging. Doing so on Amazon though is not free. Fortunately, grid researchers now can have free access to Future Grid with both Eucalyptus and Nimbus style clouds. We did have to do some tinkering to get these systems to work. First, because the user data is binary data and the communication exchange uses RPC, which may have difficulty handling binary data, it must be converted to base64 before transferring and converted back into binary data afterward. EC2 handles this transparently, if using command-line tools. Unfortunately, Eucalyptus and Nimbus do not, even though Eucalyptus is supposed to be compatible with EC2.

Furthermore, when starting an EC2 instance, networking is immediately available, whereas with Eucalyptus and Nimbus, networking often times takes more than 10 seconds after starting to be available. Thus a startup script must be prepared for networking not to be ready and hence unable to immediately download user data. The best approach to deal with this in a distribution independent manner is to wait until the primary Ethernet interface (eth0) has an IP and then continuing.

X. Related Work

Existing work that falls under the general area of desktop grids/opportunistic computing include Boinc [20], BonjourGrid [21], and PVC [15]. 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 [22], 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 VPN to transparently handle network constraints.

In the space of clouds, there exists contextualization [23]. 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 [24]. 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.

XI. 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 in Section VII: 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. Our work also makes it clear that these components do not add unnecessary overheads that are not present in statically configured grids, as shown in Section VIII. Beyond that, this paper surveys many important features of grid system and can be used as a concise overview for users interested in constructing similar systems. 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.
[4]

For future work, there are many interesting avenues to persue. The "Group Appliance" can be extended to support certificate chaining, so that users would only need to join a "GroupAppliance" and by proxy of the creater have a membership in the "GroupVPN." Many users would greatly appreciate having read/write NFS mounts, but this must be done in such a way as to maintain some level of security from potentially malicious systems. The "Grid Appliance" could potentially improve its usability 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.

[4]More traditional approaches to grids have an entity known as a virtual organization that allows sharing of resources to only select members of a grid. For simplicity purposes, our focus has been to have a flat layout, where there exists a single virtual organization maps to the entire grid. Though this does not preclude the techniques in this paper to be used in environments where this is not the case.

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] 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

[12] 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.

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

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

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

[16] 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.

[17] (2009) Amazon elastic compute cloud. http://aws.amazon.com/ec2.

[18] 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.

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

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

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

[22] 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.

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

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