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

David Isaac Wolinsky, 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**

Grid computing projects such as TeraGrid [1], Grid'5000 [2], and OpenScience Grid [3] provide researchers access to vast amounts of compute resources, but in doing so, require the adaption of their workloads to the environments provided by these systems. Researchers do not have many alternatives as creating these types of systems involve coordination of distributed systems and expertise in networking, operating systems, file systems, security, and grid middleware. This results in many research groups creating small, in-house compute clusters where scheduling is often ad-hoc, thus limiting effective resource utilization. To address these challenges, we present the "Grid Appliance." The "Grid Appliance" enables researchers to seamlessly deploy, extend, and share their systems both locally and across network domains for both small and medium scale computing grids. This paper details the design of the "Grid Appliance" and reports on experiences and lessons learned over four years of development and deployment involving wide-area grids.

I. Introduction

Grid computing presents opportunities to combine various scale, 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 or often times manage their local resources inefficiently through resource discovery and allocation by word of mouth. While there is a wealth of grid middleware available, including resource managers like Condor [4], Torque (PBS) [5], and Sun Grid Engine [6] and parallelization tools like MPICH [7], Hadoop [8], and UPC [9], most researchers see the entry barrier to installing and future management of these system as being greater than their usefulness. To address these concerns, we have implemented the "Grid Appliance" allowing users to focus on making use of grids and minimizing their effort in setting up and managing the underlying components.

At the heart of our approach lies a P2P infrastructure based upon a distributed hash table (DHT) enabling decentralized peer discovery useful for coordinating the organization of the grid. Peers are able to query the DHT with a key and potentially receive multiple values efficiently without complex searching algorithms. Network asymmetries are avoided by connecting all peers through a virtual private network (VPN) built on top of the P2P system, allowing seamless use of existing network applications. Resources are configured through files generated by a web interface, followed by automated interactions involving the DHT or VPN based IP multicast. Finally, network file systems are configured to allow users remote access to their files through both remote grid machines and their own personal resources.

The entire system has been packaged into a software repository enabling automatic configuration of physical, virtual, and even cloud resources. Users can either download preconfigured virtual machine (VM) or cloud images or configure their own through common package managers. The end result is all the same, a "Grid Appliance," a preconfigured environment emphasizing user-centricity and trivial installation. This approach allows users to focus on their tasks rather than the configuration details, providing researchers with a plug-and-play tool to create ad-hoc virtual compute clusters for their own groups, local or federated. A graphical overview of the system is illustrated in Figure 1.

The web interface used to create the appliance configuration is called "Group Appliances." At this site, users create or join groups, similar to an online social networking group. An administrators of a group has the ability to accept or deny users and remove misbehaving users. Members of a group are able to create and download configuration files, which plug into the "Grid Appliance" as a floppy disk or as a file in its file system. The file specifies the type and purpose of the "Grid Appliance" instance and uniquely identifies the owner. Upon first boot, an appliance instance contacts the "Group Appliances" site specified in the configuration file to obtain a certificate authority (CA) signed certificate, after which the system becomes completely decentralized and connects to other systems through the P2P overlay.

To justify our techniques, consider the difficulty in combining resources across disparate networks, which may or may not involve multiple research groups. Challenges such as security, connectivity, and efficiency 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 translators (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 system embraces these concerns, we assume a completely decentralized system with many if not all resources behind NATs, that users may be unfamiliar with networking considerations and managing grid organization; but the system still works well when used inside a LAN controlled by experienced IT workers.

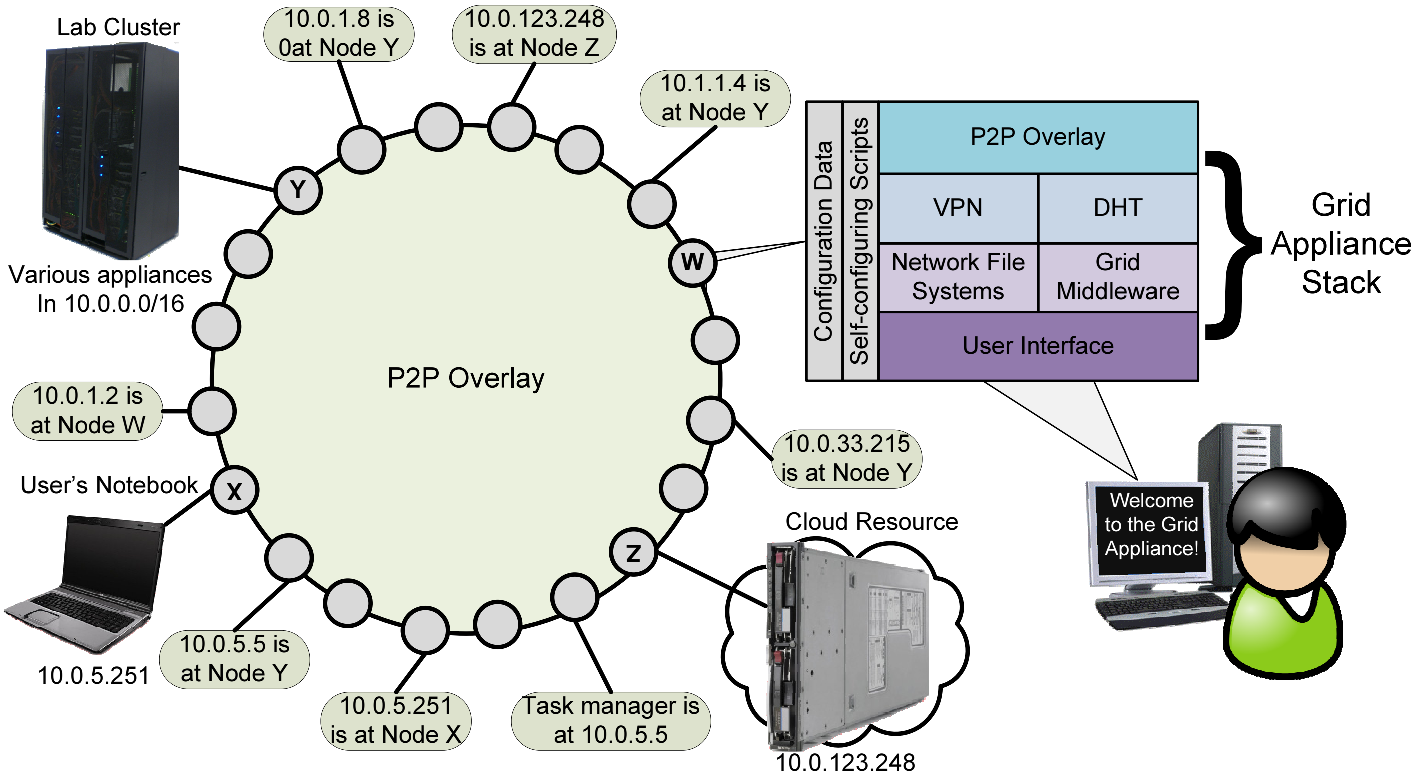

Fig. 1. A "Grid Appliance" system consisting of various users and resource types. The grid uses the P2P overlay for connectivity through a VPN and discovery mechanisms through the DHT. The Grid Appliance software stack consists of P2P software, VPN, DHT, grid middleware with self-configuring scripts,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

The rest of this paper discusses these challenges in more depth and our solutions addressing them. Section II provides an overview of the "Grid Appliance" and the systems involved as well as introduces some of our simpler contributions. In Section III, we present a detailed review of available grid middleware to address the ambiguity in the previous section. As described in Section IV, the other key component of the system is a P2P VPN that makes created distributed systems significantly more simple. Section V provides a case study of a grid deployment using standard grid deployment techniques compared to our "Grid Appliance," thus practically motivating our work. Using the system described in the previous section, Section VI reviews the overheads of self-configuration in our approach. We share our experiences from this long running project in Section VII. Finally, Section VIII compares and contrasts other solutions to these problems.

II. The "Grid Appliance"

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 V, this effort will be compared to the configuration and use of a manually configured grid.

A. Creating the Grid

The process begins with the first step, labeled "User." At this point, the user should be aware of the core user components of "The Grid Appliance," the group website, ability to deploy VMs or alternatively physical resources, and how to interact with Condor [**?**], our grid task scheduler of choice as motivated by Section III. To address users who may not be familiar with these parts, helpful tutorials are provided on www.grid-appliance.org. The process begins with a user creating a new group or joining an existing group.

There are two types of groups: "GroupVPN" groups and "GroupAppliance" groups. A "GroupVPN" group constitues a grid, 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 "GrouAppliance." As an example, consider a university with independent departments such as math, physics, computer science, and electrical engineering. All the departments may desire to share resources together, to take advantage of each others idle cycles, but when a deadline approaches, they will want priority on their contributed resources. Without priority, they could possibly setup a parallel system in addition to the "Grid Appliance" to ensure unhindered access to their resources. In this example, "GroupVPN" allows the departments to collaborate their resources into a common university grid, while still maintaining priority to their own resources through "GroupAppliances."

Create 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"

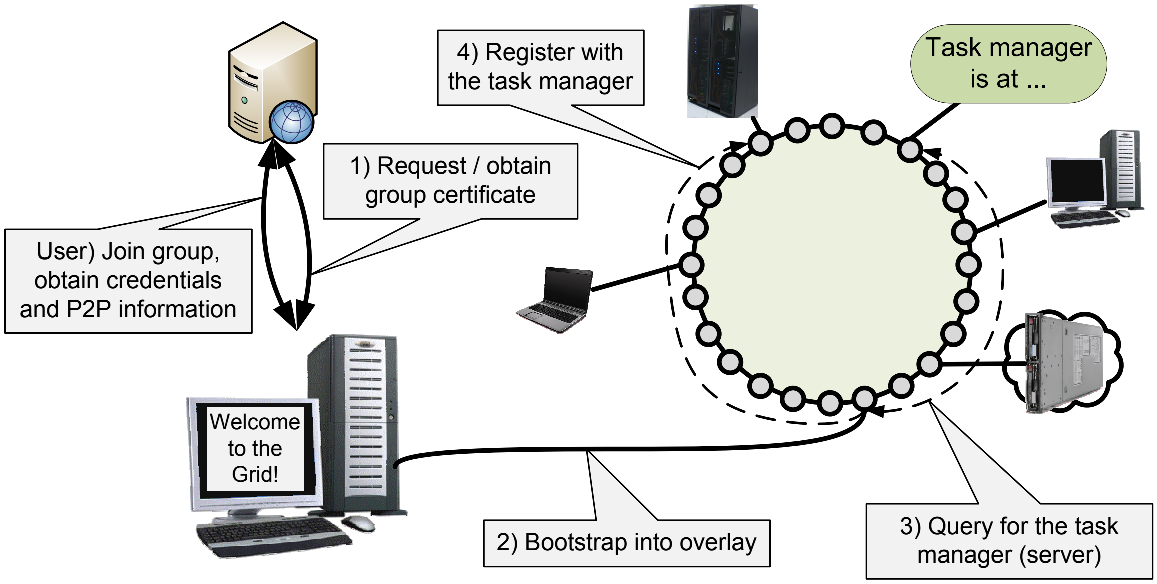
4) Register with the task manager
Task manager is at ...
1) Request / obtain group certificate
User) Join group, obtain credentials and P2P information
Welcome to the Grid!
2) Bootstrap into overlay
3) Query for the task manager (server)

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

group and then a "GroupAppliance" group matching their affiliation or creating their own. Alternatively, though not yet investigated, a user could simply join the "GroupAppliance" group, which would make them members of the "GroupVPN" by proxy through a chain of trust. Members of a "GroupAppliance" group are presented with the options of obtaining configuration files to create managers, workers, and clients.

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 perform the roles of workers, by allowing tasks to be run on them, in addition to having features useful interacting with the grid. These terms are used loosely, their use is determined by the self-configuring scripts for the middleware installed. Section III-A presents their relationship when used with Condor.

Once a user has the configuration files to bootstrap managers, workers, and clients, they can start "Grid Appliance" instances on physical, virtual, or cloud machine. Users can either download a preconfigured VM or start a cloud instance or create their own 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 with the installation of the operating system through a preseed or kickstart file. 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. Again this process can be streamlined with operating system installation or done afterward.

Following the tutorial for those interested in deploying their own grid recommends creating a single manager, worker, and 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." Thus freeing a user configuring or connecting to a grid of the worries introduced by network dynamics or constraints, issues further explored in Section IV. At this point, the machine will automatically find its place inside the grid. A manager machine will register itself with the P2P system (not shown), while clients and workers will attempt to find available managers by querying the P2P overlay, step "3," and then the managers directly, step "4." The technical details for our approach with Condor are described in Section III-A.

B. Improving User Experience

With the proceeding steps completed, the grid system has been created. A user can submit tasks and receive tasks to run on their systems. The remaining issues focus on user access. Challenges in this realm focus on limiting malicious users, while still providing features important to the local user and users in general. This raises questions like how do users easily move files across the grid or into their local machine? If a user runs a client in a virtual machine on their desktop, how can the client be informed to refuse running jobs, while the user is on the desktop?

With systems like these, there exist a lot of potential security issues and its not necessarily intuitive the best approach to make while balancing usability with security. The system must be capable of preventing a malicious user from from performing a denial of service attack on a third party on the Internet or accessing private files of a user directly connected to a "Grid Appliance," while still allowing a local user to access the Internet and their local files. Our goal is to secure the system to the point that the only viable attacks are those directly against the software running on the machine and not poorly chosen passwords or resources outside of the grid, thus creating a sandbox of sorts. To do so, tasks running on a remote machine are run by a user who has access to nothing but directories allowing that individual or those listed for everybody, they are unable to use "su" or "sudo," preventing escalation of privilege attacks due to poor passwords, and their networking is limited to the grid.

To ensure accessibility, users have access to network file systems and remote login. To secure these services, the system assumes that the first Ethernet device connects to the Internet and limits access from that network strictly to grid computing purposes and accessing the Internet. User services are prevented from use via the Internet or the VPN to prevent malicious exploitation, these servies include a SSH server and a Samba or Windows File Share with the user name set to that of the name on the group website and the password defaulted to "password." The "Grid Appliance" hosts its own Samba share as oppose to using traditional virtual machine file shares. This process is the reverse typically done by VMs, which mount the users file system into the VM. We chose to do this in reverse, to keep personal files away from the grid. To access these services, users can add a second Ethernet device, on a trusted network. In a virtual machine environment, this may be connected to "host-only" in other words, only the host machine may use it. Alternatively in a cluster, the first Ethernet may be connected to the Internet and the second to the LAN only.

When running a VM on a desktop, the VM will be unable to detect if there is an active user on the host. To address these concerns, we have written an agent that communicates with the client through the second Ethernet device. The agent discovers a server through multicast service discovery and verifies that it is running on the local machine by limiting the discovery to networks other than that of the default network device on the host. The agent notifies the client's server process, whenever a user is accessing the host. Then the server will prevent tasks from being scheduled on the local machine until sufficiently long time has passed since it last heard from the agent, typically 10 minutes.

While some grid software supports distributing data directly through it, this approach is not always the best. For example, an individual worker may only need a fraction of the data contained within a large file, transferring the entire file may make use of the grid disadvantageous. To suppor these sort of sparse data transfers, each "Grid Appliance" has a local NFS share, which is 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 called autofs. autofs works by intercepting file system calls inside a specific directory, parsing the directory link, and mounting a remote file system. In the "Grid Appliance," this occurs through the path /mnt/ganfs/hostname, where the hostname is either the IP address or hostname of another instance. Accessing a file in that path will automatically mount the NFS for that "Grid Appliance" 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. For future work, we plan on investigating means to allow selective read/write mounts based upon the user id and IP of the remote task worker.

III. Grid Middleware

Grid middleware is used to connect various distributed resources together in order to run computing tasks. These type of systems include resource management systems like Torque, Condor, and Oracle / Sun Grid Engine (SGE), which consist of three fundamental components: execute nodes, resource managers, and submission nodes. Users access a submission site, craft task description files, and submit them to a scheduler or resource manager, which will queue tasks to the various execute nodes to run when available.

Some of the issues that arise when configuring grids include: how will users connect to resources, who will be able to submit jobs, where will the job queue be located, how can priority be given to local users, how large can the grid become, and what if any changes will the user need to make to their software to run it on the grid. In Table I, we compare both popular grid solutions with recent research projects like BonjourGrid [10] and PastryGrid [**?**] that are aimed at resolving some of the more challenging issues regarding coordination.

Many grids are configured to have a single site for job submission, such that all users of the system, both local and remote, must have direct access to the submission site. At this site, users each have a unique account for their use and have the ability to execute programs and store files. This means that an administrator must explicitly create an account and that the user must be trusted. A malicious user could run an application that escalates privileges, distribute copyrighted materials, or interfere with others use of the grid. The goal of the "Grid Appliance" is to isolate users and make them self-sufficient, in doing so, we require that submission sites be created on demand and potentially be single user and optionally multiuser. By doing so, users are never given explicit accounts on privileged resources inside of secure environments. Instead all job submissions occur from their own resources.

Having a single job queue can lead to unfair sharing of resources, for example, consider a multi-site grid with a single job queue. If the managers of the job queue were in dire need for resources, they could manipulate the system in order to obtain higher priority on all resources, abusing their power and obtaining an unfair portion of all grid resources. On the other hand, a user or a site that provides resources for the grid should have high priority on their own resources, otherwise, their motivation for sharing would be limited, because they

	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, separate submission site, optionally one per user	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 collectivitely limited by the Pastry DHT	Each connected peer maintains its own job queue and submission site	None
PBS / Torque [5]	Traditional approach to dedicated grid computing	up to 20,000 CPUs[2]	Global job queue and submission site	
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

could just as well remove their resources from being shared whenever the user or a member of the site wanted to use them. Having to deal with these issues is undesirable and could inevitably lead to a fractured grid, as many users may simply desire for a simpler setup, where there is no concern about ensuring privilege on their own resources. Furthermore, when grids are financed by third parties, the third parties like to receive statistics about their use, if a member of a grid were to remove their resources every time a local user needed them, those results would not be recorded nor available to the third party. We believe at a minimum each site should have the opportunity to run their own job queue, at a minimum their will be a job queue for the entire network, and optionally, each submission site will behave as a job queue in a completely decentralized system.

While its not particularly common, some systems, like Boinc, require that a user wishing to deploy tasks on a grid compile their software using particular APIs. Some, like Condor, provide optional APIs to provide extended features like process check-pointing and migration. Having firm requirements that force users to write specialized code may work for certain environments, but doing so increases the entry barrier and may provide too much challenge for courses in grid computing. Having optional requirements, lets dedicated users take advantage of special features without affecting the use of less complex tasks.

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 and while using systems that allow cross-domain grids through middleware like Globus [11] could potentially enable these systems to meet our goals, we found the end result too complex. Boinc is completely inappropriate for our approach as it really works well when used to facilitate a single project from a central point.

A. Resource Managers and the DHT Approach

To efficiently and transparently construct wide area grids, we employ a DHT. To do so, a manager places its IP address into the DHT at the key *managers*. When workers and clients join the grid, the systems automatically query this key and configure to report to one or more managers, application

[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

dependent. Likewise, managers can query this key to learn of other managers to coordinate with each other.

Of the resource management middlewares that we have surveyed, Condor matched closest with our goals due to its decentralized properties and focus on desktop grids and voluntary computing. As described above, most of the available cluster and grid software do not easily support multiple submit points, in fact, most require another piece of software to bridge the gap between cluster and grid, or more explicitly distribution of the system.

Further motivating Condor is the ease in adding new resources. To add new resources to a Condor system, an execute or submission node must have the IP address for the manager, the rest of the system organization is performed entirely transparent to the user. Conversely, in SGE and Torque, after resources have been added into the system, the user must manually configure the manager to control them.

Finally, Condor supports 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.

A caveat to our approach is the requirement of a manager, while the system organizes through a decentralized means, it still results in a distributed system relying heavily on a small subset of nodes. For future work, we are investigating means to completely decentralized the manager node, potentially by making each node a manager for its own node and through decentralized resource discovery and priority ranking algorithms. In the meantime, we have taken advantage of a feature known as "flocking" in Condor. 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.

To configure Condor, we store managers IP addresses into the DHT using the key *managers*. When a new peer joins, it queries 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.

1) Hadoop, MPICH, and Multicast Discovery: Alternatively users may want to experiment with tools meant for LANs but not want to invest the time to install and configure them. In which case, DHT use does not translate well if they want to install software without using our appliance domain. To support these endeavours, we have investigated methods to bootstrap grid middleware through IP multicast. IP multicast works on all LANs and is used by many popular applications for discovery, such as Windows Media Center and iTunes by meanas of the UPNP (DLNA) and DAAP protocols, respectively.

The two systems that we have used to configure through IP multicast are Hadoop and MPICH. Hadoop configuration consists of a head node and worker nodes, where the head node distributes map tasks to the worker nodes. In MPICH, each resource is identically configured to support interprocess communication through the MPICH library, it is up to the application developer to determine roles of individual nodes. In both cases, the way multicast discovery works is to send out a beacon requesting that all nodes supporting these services respond.

Our appliance setup for these applications consists of a common ssh key, so that all resources in a grid can connect with each other and a multicast discovery application so a coordinator can organize the resources. The ssh key only enables access on host only networks and VPNs, preventing malicious third-parties from gaining access to the grids. The IP multicast script is run by the user on a single access node, which will act as the coordinator. The resource discovery sends out a beacon several times and after a 30 second delay, the process completes and all responding resources are automatically configured through ssh.

IV. The Motivation for VPNs

As of 2010, a majority of the Internet is connected via Internet Protocol (IP) version 4, which is quickly approaching its limit of available addresses, 2^{32} (approximately 4 billion). With the Earth's population at over 6.8 billion and each individual potentially having multiple devices with Internet connectivity, the IPv4 limitation is becoming more and more apparent. There are two approaches to addressing this issue: 1) the use of NATs to enable many machines and devices to share a single IP address but preventing bidirectional connection initiation, and 2) IPv6 which supports 2^{128} addresses. The use of NATs, as shown in Figure 3, complicates grid systems that require all-to-all communication, which include all of those which we consider. In addition, firewalls may prevent peers from receiving incoming connections. And while the eventual widespread use IPv6 may eliminate the need for address translation, it does not deal with the issue of firewalls, and the future of NATs in IPv6 is unclear.

The use of VPNs motivates beyond the impetus for traversing NATs and firewalls. With a VPN, users can avoid the headaches associated with dynamic IP addresses, as each VPN instance can claim and maintain a globally unique IP address and, regardless of the machines physical location and mobility, ideal condition for laptop users. In addition, it abstracts the user from having to be concerned about network addresses. For example, when using machines across networks or even a virtual machines inside the same LAN but behind virtual machine manager NATs, users must be wary of all nodes in the systems ability to connect with each other. When using a VPN, these considerations are unwarranted, the grid software need only concern itself of the VPN and the direct connectivity provided through it.

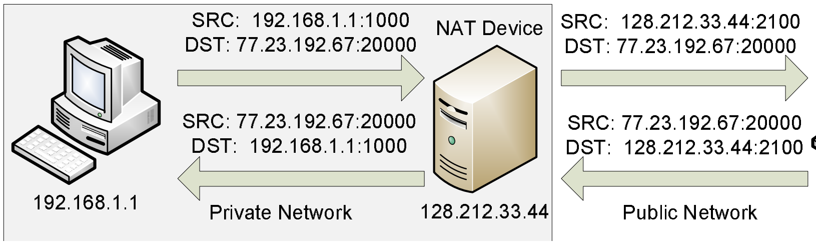

Fig. 3. A typical NAT interaction. The peer behind a NAT has a private address. When the packet is sent through the NAT, the NAT translates the source information into a public mapping, keeping the original source information so that if a packet from the remote peer comes back, it can be translated and delivered to the original source.

Our work relies on a group enabled IPOP VPN called GroupVPN [12]. IPOP through its underlying P2P infrastructure supports NAT traversal allowing peers behind NATs and firewalls to communicate directly and indirectly through relays in the P2P system. The VPN enables many of the key features in the "Grid Appliance." For example, if there were not a VPN, users of MPI and Hadoop would need to ensure that all resources were bridged to the LAN and not through a VM NAT, the typical configuration, otherwise the multicast message would not be delivered to all participants. The VPN software supports the ability to self-organize using existing infrastructures including IP multicast, public overlays, and Xmpp as described in our previous work [13]. This is in contrast to other VPNs like Hamachi [14], OpenVPN [15], Tinc [16], Violin [17], ViNe [18], or VNET [19], that are either centralized and require a dedicated node to coordinate peers or decentralized solutions that require manual configuration of links between peers.

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 "Group Appliances" 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 "Group Appliances," as its available as a VM, and host their own DHT overlay.

V. 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 4. 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."

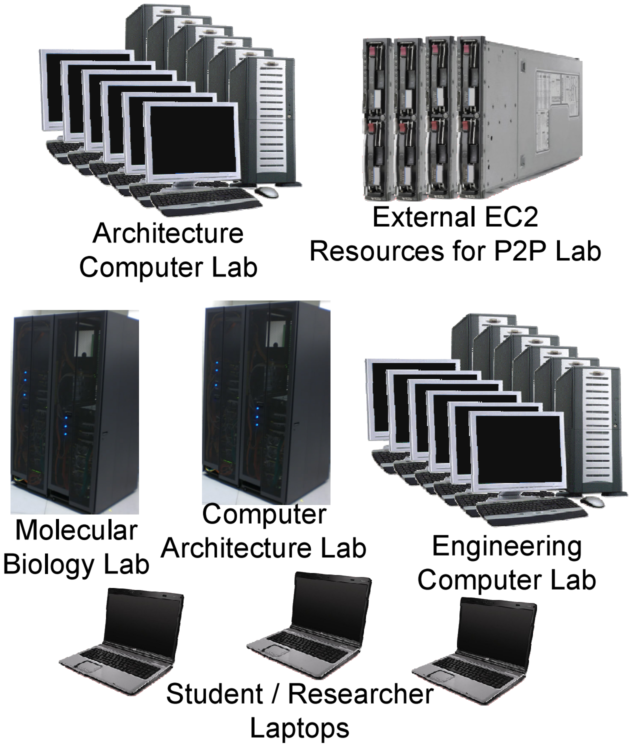
Architecture Computer Lab
External EC2 Resources for P2P Lab
Molecular Biology Lab
Computer Architecture Lab
Engineering Computer Lab
Student / Researcher Laptops

Fig. 4. A collection of various computing resources at a typical university.

A. Traditional Configuration of a Campus Grid

The first step in joining the resources 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. Thus to deal with these potential network asymmetries, the user would need to deploy a VPN. The most straight-forward approach 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.

With the grid in place, users will want to submit jobs. A global submission site will allow for a simpler though not necessarily trivial network configuration. Whereas having each user supply their own submission site would require nearly 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 personel. 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 uninterupted 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 those details have been fleshed out, 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 use a virtual machine and there are many groups that do this. Owners of the clusters could either use these virtual machines or manually install the same software on their machines.

Once the machines are fully installed, they will need to be configured to point to the global job queue or server, which in Condor 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 to 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”
```

The huge caveat is that even with this in place, if there is no mechanism for verifying the identity of the submitting node, a malicious user can obtain priorities not granted to them. 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.

A campus computing lab will lose its purpose if those directly using lab resources do not have priority over remote users. If the Condor is running natively on the host, the administrator can enable Condor features that monitor for keyboard and mouse movement that will bump jobs off the machine. Though most likely, in most common cases virtual machines will be running on the desktops. If this is the case, there is not a very intuitive solution on informing Condor running on the virtual machine, that a user is actually on the host. For the Grid Appliance, our solution was to run an agent that monitors usage on the host and reports it to the virtual machine. Discovering the existence of a virtual machine only requires that the VM have a host-only interface. The agent will send multicast discovery messages to all the VM interfaces on the host and if it discovers a VM will then send notifications of usage to a service on VM. That service will in turn simulate an active user inside the VM, causing jobs on the VM to be suspended, migrated, or removed.

Further motivating the desire for having multiple submission sites is desire for users to the ability to access their data files through NFS. While it may be easier to have a single NFS mount on all the workers, it will also add additional costs to the global submission node. Earlier we described how we use autofs to automatically mount NFS stores from submission nodes. This same approach could be used in the case of a manually configured grid. It would require that each machine was configured the same way though.

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 "Group Appliance" 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 instanteous 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 connected with the server prior to the user having access to 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 appear telling them the state of the system, including the state of 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 [**?**].

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.

VI. 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. The tasks in this evaluation are to determine the time required to start a grid individually at multiple sites individually and then cumulatively. We compare a statically configured grid versus our dynamic "Grid Appliance." 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 a Future Grid at University of Indiana's using Eucalyptus behind a AAA NAT.

Prior to beginning the evaluation a manager and submission node are started and connectivity between the two are verified. In the case of the static system, OpenVPN is run from the manager node, which has been assigned a static IP address. The experiments are run three times for each environment, the times measured during the evaluation are: "start" - time from starting the instances to when they register as being turned on, "connect" - the time delta between the end of "start" and when all resources appear in "condor_status," that is, have registered with the manager, and "run" - time taken to submit a 5 minute job to all the resources in the system, which measures the time for VPN connections to establish between the submission node and the workers. All tasks are automated through scripts with human interaction required only to start the events of grid boot and job submission. Results are presented in Figure 5.

	50 - EC2	50 - NEU	50 - UF	150 - All
Start	2:44	10:21	20:23	21:14
Connect	2:27	11:36	3:53	17:13
Run	7:15	6:35	5:53	21.19

Fig. 5. Time in minutes:seconds to start and connect execute nodes from various sites, Amazon EC2, Northeastern University, and University of Florida, to an already online resource manager, and then the time to run a 5 minute job from a freshly connected submission node.

As the systems consist of various hardware and software configurations, the time to start is only provided as a reference to potential overheads in bootstrapping the resources. Some of the interesting experiences of the experiment were: 1) the combination of the "iptables" and VMware NAT was more easily traversable than the combination of "iptables" and KVM NAT and 2) in the experiment consisting of 150 peers, nodes were actually well connected much earlier, but due to missed packets and Condor timeouts, not all resources were accounted for in Condor as early as in the other tests. With regards to the KVM NAT, it appears to be particularly aggressive as NAT mappings last for less than 10 seconds, while typical NATs keep mappings for over 30 seconds.

VII. Lessons Learned

In this section, we will present some of our previous experiences and features of Grid Appliance that proved to important lessons learned that were not obvious when we started.

A significant component of our experience stems from the computational grid provided by Archer [20], an active grid deployed for computer architecture research, available for over 3 years. Archer currently spans four seed universities contributing. The core of Archer consist of more than 500 CPUs at these seed universities as well as contributions from external users. The Archer grid has been accessed by over hundreds of students and researchers submitting jobs totaling over 200,000 hours of job execution in the past year 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 by itself, but then trying to make packages so that others can reproduces an identical configuration requires significant amounts of effort from first time package developers. To address these concerns, we provided a stackable file system also known as copy-on-write, its design was described in our previous work [21]. At this point in time the "Grid Appliance" was solely based upon VMs. Each VM would consist of 3 disks, one being the "Grid Appliance" base image, the software stack configured by us; another was something we called a module; and the last was 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.

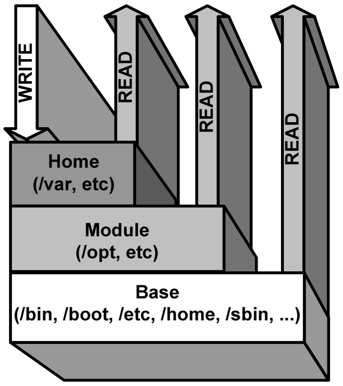

Fig. 6. Example of a stackable file system from our previous "Grid Appliance." A file will be read from the top most file system in the stack and all writes are directed to Home.

When we released new versions of the "Grid Appliance" a user could easily replace the "Grid Appliance" base image with the new one, keeping their module and home disk images. Users that wanted to extend the configuration of the "Grid Appliance" could launch into a developer mode. While in developer mode, all changes would be written to the module image and the home image was ignored. Users could run a script to prepare the module image for sharing, shutdown the VM, and redistribute the module as a package.

We actually had a few groups actually take advantage of the approach for creating modules. Though there existed a few unfortunate issues with the approach. There exists no kernel level support for stackable file systems, we had to add UnionFS [22] to the kernel, adding the weight of maintaining a kernel unto our shoulders. In addition, when we upgraded the distribution to newer versions, the module was often times broken. The last remaining issue was that the approach was incompatible with hosts other than VMs. While we still think the concept is great, we have removed it from our mainline appliance. We feel that the future work for this type of application will be to allow users to configure a system and then make it easy for them to create packages. Doing so will allow the packages to be significantly more portable and less configuration headaches.

B. Timing in Virtual Machines

A recent white paper from VMWare [23] suggests that when using virtual machines to synchronize with the hosts time through virtualized timing services and to avoid using services like NTP (network time protocol), which disagree with the host time and may have adverse affects on timing inside the virtual machine. Our experiences recommend the opposite with virtualized time responsible for drastic timing jumps and significantly off virtual clocks due to the host clocks that are incorrectly set. 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. We also had software that would not work properly since the time on the machine disagreed with the time on the remote machine, an issue when using license servers. Unfortunately we were unable to fix the timing on the host, since it was not our machine. We conclude that system developers be wary when using virtual timing and 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

When deploying VPNs, it is desirable to limit potential overlap of the VPN network space and other networks that may run the grid software. If the two overlap, the user will not be able to directly connect to the grid. Doing so would confuse the networking stack as there would be two network interfaces inside the same network address space but not connected to the same network. The best condition would be that the VPN would fail to work, but in some instances this could make the entire networking stack inoperable until the VPN was stopped. To get around this a user could potentially run the VPN behind a NAT, for example, a VM could use a VMM NAT or a cluster could be running behind a NAT device.

For the "Grid Appliance," we did not want users to have to concern themselves about these issues. To address these issues Ala Rezmerita et.al. [24] recommended using the experimental address class E ranging between 240.0.0.0 - 255.255.255.254. In order to do, Linux requires kernel modifications. At first this was not much of an issue, but over time the issues against this approach accumulated. There were both bug and security fixes for the kernel, each time requiring us to rebuild and ship that kernel to users, as opposed to using the one that came with the distribution. As we began experimenting with and had user requests to run thek "Grid Appliances" on physical and cloud machines, it became apparent that this approach was not reasonable. If a user was using a machine for more than just the "Grid Appliance," they may not want to use a non-standard kernel. The clouds did this approach in, you cannot modify the kernel in most cloud setups as they build on top of Xen, which means you are unable to provide your own kernel.

In the current model, we take advantage of the 5.0.0.0 - 5.255.255.255 address space. The advantage of this space is that like the class E it is unallocated, but it requires no changes to any operating systems. The only disadvantage is that it is extremely popular to use inside the VPN community. So that means a user would be unable to use the GroupVPN with another VPN at the same time. This is much better than having to provide kernels or worrying about users being able to not connect due to the LAN overlapping with the VPN address space. Though even with this in place, we still see users 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

In our original design, appliance would secure the VPN through the Racoon [**?**] and IPSec stack in Linux kernel. This model was kept through our first generation of Archer deployment and GroupVPN. Our realization though was that the use of IPSec only partially secured the VPN and also limited its deployment to the "Grid Appliance" or to users who were knowledgeable enough to configure an IPSec stack, which was not a requirement we wanted for those that wanted to use physical resources. With regards to securing the VPN, the IPSec solution still left the P2P overlay insecure, so while users could trust the communication, when secure connections were formed, the overlay could potentially be derailed by malicious users. Securing the P2P layer with IPSec would have been extremely complicated, since IPSec rules would need to be created for the peers prior to initiating communication and when dealing with a P2P system with dynamic members that requires clairvoyance. Still the solution would also be hampered by network configuration issues that are introduced by NATs and firewalls. IPSec was effectively not an option.

To follow this work, we implemented a security filter model that could be used to secure both VPN communication as well as 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 **??**.

E. Towards Unvirtualized Environments

As users became more familiar and comfortable with the system, they no longer want to run it inside a VM, especially if they have a dedicated computing cluster or want to run it inside the cloud. Like a VM, a cloud instance could easily be created and then shared, but a physical machine is a little more complicated as it requires specialized non-standard software to create a system and then share that image with others. Especially since other machines may have different hardware configurations that may actually prevent reusing that physical machine image.

As a result of requests from users, we focused on two means to better integrate with physical machines. First, we moved away from packaging a VM appliance and more towards creating packages. Secondly, we have improved the VPN to support a router mode whereby a single VPN can be used by many machines inside a LAN. As discussed early, the implications of packages mean that users can easily add the packages to their installation scripts for machines and have the packages automatically added. It also makes it easy to take an existing system and install software. The effect of supporting a VPN router meant that machines on 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, this was discussed more in our previous work [25].

F. Advantages and Challenges of the Cloud

Over the past year, we have had the experience of deploying the "Grid Appliance" on three different cloud stacks: Amazon's EC2 [26], Future Grid's Eucalyptus [**?**], and Future Grid's Nimbus [**?**]. All of the systems, we have encountered so far, allow a user to upload a small amount of unique data with each cloud instance started. When an instance starts, it can download the data from a static URL that is only accessible from within the instance, for example, EC2 user data is accessible at http://169.254.169.254/latest/user-data. This allows "Grid Appliance" cloud instances to be configured via user-data, which is actually the same configuration data used as the virtual and physical machines, albeit compressed. To make the supports of clouds transparent, the "Grid Appliance" goes through a process, first checking to see if there is a physical floppy disk, then if there is one in a specific directory (/opt/grid_appliance/var/floppy.img), then the EC2 / Eucalyptus URL, and finally the Nimbus URL. The process mounts the floppy, however, it was provide and then converges towards parsing the configuration data in the floppy.

Cloud systems make it very convenient to do debugging. Amazon, unfortunately, is not very cheap to do large scale testing, where starting 50 nodes for a few hours easily runs tens of dollars. If the system has bugs, this process may need to be repeated, quickly upping the cost. With the recent coming of Future Grid, we have been able to freely do this style of debugging for free. Though we have learned a little bit from using these other grid services. Because the user data is binary data and the communication exchange uses RPC, which could be mangled by some of the 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 takes more than 10 seconds after starting the VM to work. This creates a problem if the system uses a startup script to download the configuration data. There are a few potential solutions to deal with this, such as simply having a script wait until the primary Ethernet device (eth0) has an IP or alternatively have a script that hooks into the DHCP system to receive calls when a network device is configured.

VIII. Related Work

Existing work that falls under the general area of desktop grids/opportunistic computing include Boinc [27], BonjourGrid [10], and PVC [24]. 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 VPN to transparently handle network constraints.

IX. Conclusions

In this paper, we have described a novel grid architecture that enables both wide area and educational grid middleware. Our approach significantly reduces the entry barrier to constructing wide-area grids from scratch. The work required to reproduce a grid system as useful as the one constructed by the "Grid Appliance" is quite significant as illustrated in Section V. We have replaced those requirements through the use of a web interface, virtual machines, and packages.

Users begin constructing their grid by accessing either a common web interface or use our shared one to establish a group for VPN purposes and isolating the grid and subgroups to ensure priority on group contributed resources. Afterward users can deploy pre-configured cloud or virtual systems or configure physical resources by adding a package repository and installing the "Grid Appliance" packages, lending itself well to beginners but allowing more options for more power users.

Our results from Section VI show that the process of connecting the resources together is not significantly longer than that of actually starting the resources. Furthermore, we have shown that submission sites have very low overheads in connecting to the resources. The main limitation is the requirement for a global job queue or server, our future work will focus mechanisms to support completely decentralized grid computing while maintaining the trust and reliability of our current system.

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] C. Catlett, "The philosophy of teragrid: Building an open, extensible, distributed terascale facility," in *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2002, p. 8.

[2] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, and O. Richard, "Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform," in *Grid'2005 Workshop*. Seattle, USA: IEEE/ACM, November 13-14 2005.

[3] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Wrthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick, "The open science grid," in *Journal of Physics: Conference Series*, vol. 78, no. 1, 2007, p. 012057.

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

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

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

[7] A. N. Laboratory. (2010, July) MPICH2. http://www.mcs.anl.gov/research/projects/mpich2/.

[8] Apache. (2010, July) Hadoop. http://hadoop.apache.org/.

[9] T. El-Ghazawi, W. W. Carlson, and J. M. Draper. (2003, October) UPC language specification v1.1.1.

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

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

[12] D. I. Wolinsky and et al., "On the design and implementation of structured P2P VPNs," in *ARXIV 1001.2575*, 2010.

[13] D. I. Wolinsky, P. St. Juste, P. O. Boykin, and R. Figueiredo, "Addressing the P2P bootstrap problem for small overlay networks," in *10th IEEE International Conference on Peer-to-Peer Computing*, 2010.

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

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

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

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

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

[19] 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.

[20] 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

[21] 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.

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

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

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

[25] 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.

[26] (2009) Amazon elastic compute cloud. http://aws.amazon.com/ec2.

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

[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.