On the Use of Group-Oriented Autonomic VPNs for High-Throughput Computing

David Isaac Wolinsky and Renato Figueiredo
University of Florida

ABSTRACT

High-throughput computing (HTC) attempts to acquire and use as many resources, both dedicated and opportunistic, as possible. In this paper, we explore the challenge of connecting resources across administrative domains from different institutions and into users' homes. Several works discuss the usage of virtualization and middleware as means to connect and secure resources. In this paper, we present methods that automate the configuration of resources through group-based Web 2.0 interfaces and connect the resources through a peer-to-peer (P2P) based virtual private network (VPN) creating secure ad-hoc, decentralized, and distributed computing grids supporting both experts and non-experts alike. We verify our approach quantitatively by using our reference implementation to connect resources distributed across institutions and compute clouds to determine the time for the system to become fully active and for a single user to submit and complete jobs on all the resources.

1. INTRODUCTION

High throughput computing (HTC) is a form of opportunistic computing that, unlike high performance computing (HPC) seeking only powerful resources, benefits from ad-hoc, arbitrary resources including computers found in computer labs, homes, and offices as well as cloud resources and retired HPC clusters. Creating and maintaining HTC systems that cross administrative domains (grids) require expertise in networks, operating systems, and grid middleware to configure the sites uniformly and guarantee connectivity amongst sites involved.

Administrators each have their own way of configuring resources and may be hesitant or unwilling to configure resources in a way that conflicts with the environmental norm. This conflicts with the requirements of merging resources across administrative domains as computing resources are typically configured uniformly having a common environment and network rules. Administrators would prefer not having exceptional rules for a subset of resources under their domain and do not want to grant access to lesser known, remote users. Network constraints such as firewalls and network address translation (NAT) can prevent cross domain communication. Though this may be amerliorated by exceptions, additional rules may be required for each new cluster or resource added to the cross domain grid.

HTC clusters are not limited to large systems requiring dedicated administrators, there are many systems that discuss the use of desktop or opportunistic grids, like Boinc [1] and PVC [9]. While Boinc solutions may be easily configured, the approach relies on a centralized scheduler and that Boinc applications be embarrassingly parallel. While PVC provides enables parallel tasks and more decentralized system configuration, the approach has scalability concerns and relies on a centralized node to assist in node connectivity. In general, there exist no solutions that provide scalable, self-configuring, decentralized grid systems for non-experts.

Connecting resources distributed across the Internet can be challenging due to limit of IP (Internet Protocol) addresses available to an organization. NAT further complicates the issue by limiting the formation of direct links between remote sites without external assistance. When creating HTC systems across a small set of institutions approaches that use Internet Service Provider VPNs, Layer 2 Tunneling Protocol (L2TP) VPNs, and other user-configured VPN approaches can be used. Most solutions require some form of centralization and static links, as systems expand dynamically, the manual configuration of the system grows significantly. VPNs do, however, provide means of securing the system and, through the use of proper middleware, can allow users to interact with each others resources without additional configuration from the local site administrator.

The Archer project [3], a collaborative academic environment, links institutions and external users for the purpose of computer achitecture research provides a real system having the described constraints. Many researchers have a need for resources occassionally and rather than investing in a large pool of dedicated resources for a single institution they are able to pool their resources together using HTC mechanisms. The resulting system allows individuals the opportunity to complete their jobs quickly and ensure that their resource contribution is not idle when locally unused. This use case potentially introduces a new issue where the users may not be experts nor have the ability to include an expert in the construction of the system.

Explicitly the requirements for a system in these environments are: 1) users should be able to easily add and remove resources, 2) resources should not require configuration to allow remote users access, 3) tasks that run inside the grid should not have access to external resources, 4) external resources should not be able to access the grid, 5) resource priority should be granted to the resource's owner, and 6) malicious users should be able to be removed. In this paper, we describe our approach to handling these requirements to enable the creation of a dynamic, decentralized HTC grid through a novel approach involving decentralized overlays enabling a self-configuring VPN and HTC environment.

In previous work, we bundled IPOP [4] for decentralized virtual networking and other grid essentials into a virtual

machine called the Grid Appliance [13]. The approach was highly coupled and while it made it easy to connect to existing grids to add or access resources, it required expertise for users to create and manage their own independent grids. This paper extends that work to enable non-users to create and manage their own grids through a group-oriented model embodied in a web 2.0 infrastructure providing a public key infrastructure along with VPN and grid configuration. The approach describes methods that can be used by non-experts to configure and combine resources from virtual, physical, and cloud environments.

The rest of this paper is organized as follows. Section 2 surveys core grid middleware, while Section **??** describes the methods by which we enable user-friendly creation and management of grids. A qualitative comparison of our system to other appraoches is presented in Section 4. We perform quantitative evaluations on our system and present our findings in Section 5. Finally, in Section 6, we conclude with a discussion on real systems using our approach.

2. CORE GRID MIDDLEWARE

This section surveys existing grid middleware solutions presenting existing software solutions that address the constraints of our system.

2.1 Task Schedulers

The most fundamental requirement of a cluster is the task scheduler, each task scheduler has a general focus and selecting one that works well in a specific environment can make the configuration of the system significantly easier. Generally, there are three approaches to configure HTC clusters: 1) task workers pull from a centralized manager as employed by Boinc [1], 2) task workers receive jobs from a centralized submission cite, and 3) task workers receive jobs from any member of the HTC infrastructure. Both 2 and 3 can be implemented using a myriad of different job schedulers with verifying levels of difficulty. Task schedulers supporting this behavior include Condor [7], Sun Grid Engine [10], and PBS and its relative Torque [2].

By analyzing our first requirement, we quickly realized that by default none of the task schedulers, besides Condor, could support an environment that did not have a centralized user list. While Condor does support similar behavior, it has separated the components that keep track of resources and negotiates resource allocation from those that make the resource requests and submit tasks. This abstraction allows for a simple centralized system to maintain the grid without requiring any runtime configuration. In addition, Condor allows open unauthenticated access to the grid as long as a peer is within a subnet, which is where a VPN comes into place ensuring that only members of the VPN have access to grid resources. Note though that other task schedulers could potentially provide this feature through modification or additional middleware, like Globus [5]. This amongst other reasons, such as it being free, having an active community, and focus on opportunistic cycles, led us to select Condor for our task scheduler. In the list of requirements, Condor also handles the ability to assign groups or institutions privilege on their own resources when shared in a collaborative environment. Though in this selection, there still remains the issue of enabling multiple collaborative resource negotiators so that a single site going offline does not prevent new tasks from running on the grid.

2.2 P2P VPNs

Many of the requirements made for our system can be solved through the use of a VPN. A VPN can assist in dealing with connecting network constrained resources, securing the grid from the outside world, and removing malicious forces inside. Grid computing has seen its fair share of VPNs such as ViNe [12], Violin [6], and VNET [11]. At issue with these approaches is their lack of self-configuration, namely that static links between peers must be created and security credentials must be manually distributed. Additionally, none of the approaches support direct connectivity between NAT and firewalled peers without additional configuration from the user limiting their applicability for such resources to communicate with each other through proxy servers. In PVC [9], the authors describe an approach that self-configures through a centralized server with NAT traversal support, which works on approximately 60% of NAT devices and only when used without a stateful firewalls. The approach has drawbacks with respect to the requirements for this system: the use of a centralized key distribution center (KDC) and lack of encrypted links.

IPOP [4] provides a P2P virtual network with decentralized and self-configuring link creation with NAT traversal support that works with approximately 90% of NATs using a distributed hash table (DHT) for address allocation and resolution. Prevously, approaches [13] used IPsec for security or went without it entirely as IPOP lacks the ability to secure links. In [14], we presented GroupVPN, which transforms IPOP into an automated VPN with enhanced NAT handling through the use of decentralized relays (proxies), enabling two hop, low latency connections when NAT and firewall traversal fails.

To implement authentication in the system, we employed a public key infrastructure (PKI), which we made accessible through a group based Web 2.0 environment. Users were able to create and join groups, and group owners are able allow users to join, promote users to administrative capabilities, or remove users who have overstayed their welcome. A PKI has a very natural P2P aspect, in that, peers can mutually authenticate each other by verifying signatures on the exchanged certificates unlike the centralized authentication implemented in a KDC. To automate this, users download a GroupVPN configuration file through the group website. Upon configuring the GroupVPN with the configuration file through a command-line application, the GroupVPN will automatically obtain a signed certificate the next start. To obtain the signed certificate, the GroupVPN sends a certificate request to the group server through HTTPS and provides a shared secret contained in the configuration file, this uniquely authenticates the user and if the user has been given permission, the server will sign the certificate request. Removal of peers is performed in multiple ways, the simplest and most reliable is a user revocation list located at the group website, additionally, the system supports decentralized revocation a broadcast and distributed data stores.

3. IMPLEMENTATION AND DESIGN

Middleware alone does not make a grid, the system must be configured for the components to work with each other. This section discusses a solution to this issue using a group infrastructure to create and manage a grid with packaged environments allowing users to configure their own resources independent of pre-existing virtual machine images.

3.1 GroupAppliances

An appliance is defined as a dedicated, blackbox device requiring little if any configuration from the user. While traditional computing appliances like a router, network storage, or even cluster resources have been available as hardware appliances, the recent resurgence of virtualization initiated by VMware and Xen has made software appliances by means of virtual appliances popular. Recently, cloud computing has become popular in large part thanks to Amazon's EC2. Both virtual and cloud resources present themselves as cheap computing for HTC and opportunistic computing purposes, because they can be setup in such a way as to have no or limited effect on users' computers and can be shutdown when no longer in use. Though users have released both virtual and cloud appliances to tap into these resources, they still require configuration from the user so that they can connect with each other and to form a grid and the packaged solutions cannot easily be applied to hardware resources.

Our solution is the creation of a generic software stack that self-configures based upon a user input configuration file. The contents of a configuration file are the type of resource (dedicated compute node, job scheduling node, a mixture of the two, or the job negotiating central server); the user's group and username on the site; and the grid's GroupVPN configuration data. The configuration file is generated from a Web 2.0 group based infrastructure called GroupAppliance, using a single GroupVPN group to connect all members of the grid together in a VPN but using the GroupAppliance group to distinguish their resource contributions. Thus many GroupAppliances groups can be linked together through a GroupVPN group. By distingushing resource contributions, users are able to get credit and gain priority to their resources when submitting tasks to run.

When a user downloads the configuration file from the GroupAppliance infrastructure, the data is stored in a floppy disk image that can be used on physical resources by writing the image to a real floppy disk or to a USB drive, a VM by adding a virtual disk image to the VM, and clouds through instance specific configuration data. EC2 provides per-cloud instance configuration in a parameter they call "user data" that allows a user provide up to 16 KB of data that will be available only to that cloud instance. At the time of this writing, it appears that EC2 is the only cloud provider to offer this option. Alternatively users could configure an image specifically to run for this cloud by inserting the floppy image into the cloud image and then generating cloud instances from this new image or the user could setup a storage cloud where the cloud instances could retrieve the floppy. Because the floppy contains private GroupVPN configuration data it cannot be stored on public resources.

Upon booting, the grid configuration scripts parses the floppy to determine how the machine will self-configure. Task schedulers will insert an ad into the DHT, whereas resources and task submitters query the DHT for the list of schedulers. If there exists more than one scheduler, the resources and submitters will use Condor flocking to access the remaining resources.

3.2 Constructing Environments

Often VMs are favored for the distribution of complicated applications as experts can configure them and release the results as a complete working system. Though this prevents limits non-experts to the VM appliance, which may be undesirable for users that want to configure their own systems without reuse of the existing VM. While guides may exist for the creation of systems, the systems are too complex for non-experts to digest and produce similar results. Alternatively, we have moved towards the use of packages (DEB and RPM) that can be installed in arbitrary environments and through the use of package managers (APT and YUM) handle configuration such that the requirements listed in the introduction are handled. Packages can be provided that automatically install and configure the task scheduling middleware and a VPN as well as sandbox the environment limiting users to accessing the virtual network and not external resources such as other local resources and the Internet. The remaining components are configurable through the GroupAppliances interface and decentralized through the DHT.

The most important components in securing an environment are limiting internal and external access from inside the system. Specifically, interal resources should avoid password enabled accounts to avoid cases where users submit tasks that attempt to provide more priviledged access to the user. In the event that this is not possible, ensuring that the task scheduler runs as user without the ability to switch permissions can prevent this attack, for example, preventing *su* access in Linux. By default, Condor will run jobs as either nobody or the user named "Condor". This limits access to many of the core components of the system already, but it does not limit the users ability to read files that allow reading from anyone on the system and the ability to communicate to external resources from inside the machine. To prevent reading personal data, simply making the user data directory readable by the user and group who own the files and directories and not by others. By configuring the firewall to only allow the Condor user to only have the ability to send packets over the virtual network device.

For job submission nodes, there is one more consideration, we attempt to make these as user-friendly as possible. To do this, we have enabled file system and remote access capabilities for the machines so that a user on the same host can access the resources without having to configure additional utilities or using the VMs interface, if available. Though, we do not want users to access the data through the virtual network or the Internet from a public address, with regard to virtual machines, we create a second network card that supports host only connectivity. By binding all user applications to use the network interface, they do not require extra security enhancements. Alternatively, applications like SSH could be enabled to only allow private key based login. In our model, we expect only dedicated compute nodes and possibly the job negotiator running on physical or cloud resources and user nodes to use VMs.

To prepare a system is straight-forward, users configure the package manager to link to a package distribution site and then install the desired packages for the grid resource. When finished, the user can restart the device or restart the grid service with the floppy image adding a new resource to the grid. The VPN will acquire a signed certificate and grid configuration scripts will configure Condor and other services through interaction with the DHT.

4. RELATED WORK

There are many projects that seek to provide easy to use resources for HTC and opportunistic computing. We focus on two approaches whose focus is user-friendly dynamic

grids: PVC [9] and GPU [8].

PVC or Private Virtual Clusters creates instant grids using a PVC specific virtual network and task scheduler as well as VMs to isolate remote jobs from users' resources. Resources discover and TCP NAT traversal are performed through a centralized system broker, though it is unclear how the resources are configured with the knowledge of the broker, nor how a broker is configured. Loss of the broker can prevent usability of the system. PVC virtual network lacks privacy, links are authenticated through a KDC but messages are not encrypted or authenticated. PVC scalability constraints are unclear, as experiments were limited to 8 nodes. In contrast, our approach focuses on privacy, scalability, and self-configuration through a decentralized system.

GPU or Global Processing Unit uses the Gnutella P2P system to create a completely decentralized computing grid. The authors state that the expected size for grids range from 5 to 15 nodes. While the authors do not mention NAT traversal, there are many Gnutella systems that do support various forms of it. There is no mention of providing safety to the users' resources from malicious tasks. While GPU provides easy configurability, it lacks the ability to run jobs in a sandbox and support large pools of resources.

There are many other desktop grid environments that use Boinc as the underlying method to push jobs. As we explained earlier, Boinc uses a few approaches that we found undesirable for our requirements with the primary issue that Boinc job scheduling is heavily centralized. In addition, for Boinc systems that allow running arbitrary applications, it is unclear how secure they are.

5. EVALUATION

This evaluation focuses on the time required to create and utilize a grid consisting of various distributed resources built a reference implementation of the system described in this paper. Using VMware resources behind a Cisco and "iptables" NAT at the University of Florida, KVM resources behind an "iptables" NAT at Northeastern University, and Cloud resources provided by EC2, pools of 50 resources from each site were booted independently and then in parallel. The resource connected with an existing GroupVPN and GroupAppliance group with a negotiator and client node already online. Upon all the resources connecting, the client submits a job to all resource. Time for the machines to connect starts when the resources begin their deployment (copying of files and starting the VMs) and finishes when all of the deployed resources appear in "condor_status". Time for connectivity begins when a client client submits a job to sleep for 5 minutes to every resource and all tasks have completed. Results are presented in Figure **??**.

	EC2	NEU	UF	All
Start	0	0	0	0
Connect	0	0	0	0

Table 1: Time in seconds required to start and connect to various grid deployments using the stack described by the paper.

6. CONCLUSION AND REAL USE CASES

In this paper, we presented a novel approach to creating autonomic grid systems through a configuration data generated and distributed through a user-friendly Web 2.0 group-based infrastructure providing configuration for both the grid and the VPN. The VPN provides a completely decentralized system having no single point of failure. Grid resources are pre-configured with only identity and discover each other through the VPN's DHT. Users never need to access a console to create a HTC cluster, an attractive feature for non-experts.

We have several deployments using the system described in this paper. Over the past 2 years, we have had an active grid deployed for computer architecture research, Archer [3]. Archer currently spans four universities with 500 resources, we have had hundreds of students and researchers submitting jobs with over 150,000 hours of total job execution in the past year alone. Groups at the Universities of Florida, Clemson, Arkansas, and Northern Switzerland have used it as a tool to teach grid computing. Purdue is constructing a large campus grid using GroupVPN to connect resources together. Recently, a grid at La Jolla Institute for Allergy and Immunology went live with minimal communication with us.

References

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

[2] Cluster Resources. Torque resource manager. `http://www.clusterresources.com/pages/products/torque-resource-manager.%php`, March 2007.

[3] R. Figueiredo and et al. Archer: A community distributed computing infrastructure for computer architecture research and education. In *CollaborateCom 2008*.

[4] 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.

[5] Globus Alliance. Globus toolkit. `http://www.globus.org/toolkit/`, March 2007.

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

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

[8] T. Mengotti, W. Petersen, and the GPU Team. GPU: a global processing unit. `http://gpu.sourceforge.net`, January 2010.

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

[10] Sun. gridengine. `http://gridengine.sunsource.net/`, March 2007.

[11] 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.

[12] M. Tsugawa and J. Fortes. A virtual network (vine) architecture for grid computing. *International Parallel and Distributed Processing Symposium*, 0:123, 2006.

[13] D. Wolinsky and R. Figueiredo. Simplifying resource sharing in voluntary grid computing with the grid appliance. In *2nd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid 2008)*, 2008.

[14] D. I. Wolinsky, L. Abraham, K. Lee, Y. Liu, J. Xu, P. O. Boykin, and R. Figueiredo. On the design and implementation of structured P2P VPNs. In *ARXIV 1001.2575*, 2010.