On the Use of Autonomic Groups for Distributed, Decentralized, Dynamic 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 as possible, both dedicated and opportunistic computing cycles. 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), which seeks 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. The two most significant challenges in creating a HTC clusters across administrative domains are 1) restricting remote user access to executing in a doubly secure environment that prevents access to external resources from inside the cluster and internal resources from outside the cluster and 2) handling firewalls and network address translation (NAT) that prevent direct communication between peers in the cluster.

Administrators only allow users who have permission to access resources. In the HTC realm, the burden this places upon administrators varies with the type of environment and the middleware being used. Commonly, each user has an account to access the resources directly or through a central service provider, requiring the administrator create an account and enforce quotas on disk, memory, and processor usage. When a problem arrises, it is the responsibility of the administrator to contact both the individual and individual's organization. Ideally resources should be configured in such a way to sandbox the environment to eliminate or at least limit potential user mishaps.

Connecting resources distributed across the Internet can be challenging due to limit of available IP (Internet Protocol) addresses available to an organization, which has been ameliorated by network address translation (NAT), though NAT devices and firewalls create issues in forming direct connectivity between remote sites. When creating HTC systems across a few institutions approaches that use Internet Service Provider VPNs, Layer 2 Tunneling Protocol (L2TP) VPNs, and other user-configured VPN approaches can be used. Though because these require centralization and static links, as systems expand dynamically, the cost of manually configuring these environments are no longer reasonable.

The architecture constraints described can exist in real systems such as a collaborative academic environment linking institutions and external users. Many researchers have a need for resources occassionally and rather than investing in a large pool of dedicated resources for a single institution they could 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.

Explicity the requirements that we set for such systems are 1) users should be able to connect to the grids using their own resources so that no administrators need to maintain operating system accounts for users, 2) resources can communicate with each other with no configuration from the users regardless of the network constraints, 3) tasks that run inside the grid do not have network access to external resources, 4) external resources do not have network access to the grid, 5) users should have priority on their institutions and own contributed resources, 6) malicious users can be removed and at least confined to only using their own resources, and 7) resources should be able to be easy to add and remove. In this paper, we describe our approach to handling these requirements to enable enable the creation of an ad-hoc, dynamic, decentralized HTC grids.

Our contributions in this paper focus on a novel self-configuring grid environment through the use of a group based Web 2.0 environment that can connect distributed physical, virtual, and cloud resources through a P2P-based VPN that traverses NATs and firewalls without user configuration. The use generic physical and cloud resources makes our approach unique in that it is not limited to preconfigured virtual machines (VMs), but also allow users to easily create their own environments. To do this, we have released our system as a set of software packages (e.g., deb, rpm), which can be easily installed through package managers (e.g., APT, Yum) by non-experts in a matter of minutes. All would be for naught without a task scheduler, as such we compare some popular task schedulers against our requirements.

The rest of this paper is organized as follows. Section 2 describes the architecture of our system. A qualitative comparison of our system to other appraoches is presented in Section 3. We perform quantitative evaluations on our system and present our findings in Section 4. Finally, in Section 5, we conclude with a discussion on real systems using our approach.

2. OUR SYSTEM

In this section, we describe the main components of our system and how they come together to provide decentralized grid computing systems. Prior to describing our system, the critical component the binds the system together is the task scheduling middleware as much of the systems configuration depends on what it provides. Afterwards, we present our contributions including the the group environment, P2P VPN, and creating appliances through package management.

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 [4], 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 [8], Sun Grid Engine [12], and PBS [3, 2] and its relative Torque [10].

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 alocation 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. This is where the importance of a VPN comes into place, to ensure 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 [1]. 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 [14], Violin [7], and VNET [13]. 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 [11], the authors describe another approach that is more self-configuring than other approaches and has NAT traversal support though it is limited to approximately 60% of NATs though it does not work on stateful firewalls, the drawbacks of the approach with respect to our requirements are that it requires a centralized key distribution center (KDC) for verification, does not support encrypted links, and is not publicly available for use.

One promising approach is IPOP [6], though IPOP lacks the ability to secure links, it provides a P2P virtual network with decentralized and self-configuring link creation with NAT traversal support that works with approximately 90% of NATs. IPOP uses a distributed data store embeded in the P2P infrastructure known as a distributed hash table (DHT) to assist in address allocation and resolution. In [15], we presented GroupVPN, which transforms IPOP into an automated VPN with enhanced NAT handling through the use of decentralized relays (proxies) in IPOP, 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 with each other by verifying the signatures on the exchanged certificates unlike the use of a KDC in PVC that requires every user to authenticate through a middleman prior to communicating with each other. 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.

2.3 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 has made software appliances by means of virtual appliances popular. Soon thereafter, cloud computing has become popular in large part thanks to Amazon and EC2. Both virtual and cloud resources present themselves as cheap computing for HTC and opportunistic computing purposes, because both 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 such as dedicated compute node, job scheduling node, mixed, or the job negotiating central server; the users 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.

2.4 Constructing Environments

Most approaches in this field are usually application or environment specific and while they provide a recipe for creating these environments they do not provide guides that non-experts can digest to produce similar results. Often VMs are favored for the distribution of complicated applications as experts can configure them and release the results as a complete working system. We argue for the use of packages that can take arbitrary environments and ensure that the requirements we have listed in the introduction are handled. Specifically an environment should have the task scheduling middleware, a VPN, be sandboxed such that users are limited to accessing the virtual network and not external resources such as other local resources and the Internet.

The most important components in securing an environment are limiting internal and external access from inside the system. Specifically, when speaking about internally, we recommend that users do not create password enabled accounts on available resources. If this is unavoidable, then ensure that the user that the task scheduler runs as has no ability to switch user permissions, such as in Linux by preventing them from using *su*. 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, simplying 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. A little more goes on behind the scenes. In order for the system to self-organize knowing only the generic group information, we rely on IPOP's DHT. Negotiators place ads in the data store and client and worker nodes query the DHT to discover the negotiator for their site.

3. RELATED WORK

There are many projects that seek to provide easy to user resources for HTC and opportunistic computing. We focus on two approaches whose focus is user-friendly dynamic grids: PVC [11] and GPU [9].

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 each other through a system broker, though it is unclear how the resources are configured with the knowl-

edge of the broker, nor how a broker is configured. Peers use the broker to form direct communication links with each other through TCP NAT traversal. Due to the requirement of a broker at least one machine must be public addressable and if the node goes offline, it could prevent resources from communicating with each other. Also PVC does not provide a VPN but rather a VN, since the links between peers are not encrypted, though they are authenticated through the broker in a form of KDC. It is unclear what the node constraints in a PVC system are, though experiments are limited to 8 nodes. PVC has not been released to the public in source or binary format. In general, the PVC paper is unclear on the scalability and self-configuring of the approach. Though with focus on the broker, it appears that our approach is significantly more decentralized. Also our approach is both available in source and binary formats.

GPU or Global Processing Unit uses the Gnutella P2P system to create a completely decentralized computing grid that runs on the contributors system. 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.

4. EVALUATION

In this evaluation, we focus on the time required to create varying size distributed grids. To start the experiment, we begin with an existing GroupVPN and GroupAppliance group with the configuration floppy ready as well as an already booted negotiator and client node. Then we boot varying sizes of worker nodes in three different environments and execute a job that requires them to sleep for 5 minutes. This experiment shows how long it takes resources to become connected to the negotiator and then the client scheduling machine. To calculate the time to become connected with the negotiator, we wait for condor to officially recognize all resources through "condor_status". To calculate the time to become connected with the client, we subtract 5 minutes from the completion of the last task.

The environment we have chosen are EC2, University of Florida, and North Eastern University. We chose EC2 as a representation of adding cloud resources and that all the resources have public IP addresses as such there will be no need for NAT traversal. At the University of Florida, all resources will run as virtual machines and are behind 2 layers of NAT devices. At North Eastern University all resources are behind a single "iptables" based NAT. We boot from 1, 5, 10, 20, 25, and 50 resources in each environment 5 times to obtain our values. The time starts as soon as the scripts to deploy the resources are executed and conclude when all resources show in "condor_status". Therefore this includes the time it takes to copy the VMs to each host, start the VMs, connect to the VPN, and finally with Condor. Our results are presented in **??**.

5. CONCLUSION AND REAL USE CASES

In this paper, we presented a novel approach to creating distributed, decentralized, and dynamic grids. To make the environments self-configuring, we rely on configuration data distributed through a user-friendly Web 2.0 group based infrastructure that provides configuration for the grid and VPN. The VPN provides a completely decentralized system that has no single point of failure. The one lacking feature of our approach is the single job negotiator, we are actively working with the Condor group to improve this approach. The key component that makes our approach attractive for non-experts is that they never need access a console to create a HTC cluster for themselves.

We have several real systems using all or components of the features described in this paper. Over the past 2 years, we have had an active grid deployed for computer architecture research, Archer [5]. Archer currently spans four universities with 500 resources, we have had 100s 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 and Clemson as well as a Swiss university have used it as a tool to teach grid computing. Purdue is planning a large campus grid that will use the GroupVPN to connect resources together. Most recently, a grid at La Jolla Institute for Allergy and Immunology went live with minimal communication with us.

References

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

[2] Altair. Openpbs. `http://www.openpbs.org/`, March 2007.

[3] Altair. Pbs pro. `http://www.altair.com/software/pbspro.htm`, March 2007.

[4] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*.

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

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

[7] X. Jiang and D. Xu. Violin: Virtual internetworking on overlay. In *In Proc. of the 2nd Intl. Symp. on Parallel and Distributed Processing and Applications*, pages 937–946, 2003.

[8] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

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

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

[11] A. Rezmerita, T. Morlier, V. Neri, and F. Cappello. Private virtual cluster: Infrastructure and protocol for instant grids. In *Euro-Par '06*.

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

[13] A. I. Sundararaj and P. A. Dinda. Towards virtual networks for virtual machine grid computing. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And*

Technology Symposium, pages 14–14. USENIX Association, 2004.

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

[15] 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 *TR-ACIS-09-003*, October 2009.