5-2010

Building Scientific Clouds: The Distributed, Peer-to-Peer Approach

Linton Vadakedathu
Clemson University, lintonab.cs@gmail.com

Building Scientific Clouds: The Distributed, Peer-to-Peer Approach

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

by
Linton Abraham Vadakedathu
May 2010

Abstract

The Scientific community is constantly growing in size. The increase in personnel number and projects have resulted in the requirement of large amounts of storage, CPU power and other computing resources. It has also become necessary to acquire these resources in an affordable manner that is sensitive to work loads. In this thesis, the author presents a novel approach that provides the communication platform that will support such large scale scientific projects. These resources could be difficult to acquire due to NATs, firewalls and other site-based restrictions and policies. Methods used to overcome these hurdles have been discussed in detail along with other advantages of using such a system, which include: increased availability of necessary computing infrastructure; increased grid resource utilization; reduced user dependability; reduced job execution time. Experiments conducted included local infrastructure on the Clemson University Campus as well as resources provided by other federated grid sites.

Dedication

This thesis is dedicated to my family who stood by me with love and support. It is also dedicated to Dr. Sebastien Goasguen and the School of Computing at Clemson University.

Acknowledgments

The Author would like to acknowledge Dr. Sebastien Goasguen for his guidance and support in all research and course work done during the Masters program. The author thanks Michael Murphy and Michael Fenn for their time and support. The author acknowledges the guidance of Dr. James Martin and Dr. Pradip Srimani towards the correction and completion of this thesis. Finally, the author would like to acknowledge David Wolinsky of the University of Florida for his assistance in various research related undertakings.

Contents

Title Page . . . . **i**

Abstract . . . . **ii**

Dedication . . . . **iii**

Acknowledgments . . . . **iv**

List of Tables . . . . **vii**

List of Figures . . . . **viii**

List of Listings . . . . **ix**

1 Introduction **1**
- 1.1 Research Questions 3
- 1.2 Motivation for Research 3
- 1.3 Distributed Computing 4
- 1.4 Virtualization and Cloud Computing 5
- 1.5 High Throughput Computing and Condor 8
  - 1.5.1 Condor at Clemson 10
- 1.6 Peer to Peer Architecture vs. Client-Server Architecture 10
- 1.7 Distributed Hash Tables 12
- 1.8 From P2P to DHT to Brunet, then came - IP Over P2P (IPOP) 15
  - 1.8.1 Architectural Overview 17
- 1.9 Virtual Organization Cluster (VOC) 20
  - 1.9.1 Physical Administrative Domain 22
  - 1.9.2 Virtual Administrative Domain 23
  - 1.9.3 Provisioning and Execution of Virtual Machines 24

2 Related Work **25**
- 2.1 Chord 25
- 2.2 Pastry 26
- 2.3 Kademlia 27
- 2.4 Brunet (Based on Symphony) 28
- 2.5 Work Related to VOCs 29

3 Proposed System **31**
- 3.1 Overlay Networking Details 31
- 3.2 IPOP Bootstraps and Namespaces in Clemson 34
- 3.3 Multi-site communication and Watchdog 38
- 3.4 Condor IPOP Pool - Local/Single Site 39

3.5 Condor IPOP Pool - Multiple Grid Sites 42

4 Experiments & Results 44

4.1 Building the Physical and Virtual Clusters 44

4.2 Local Testing 45

4.2.1 Experiments 45

4.2.2 Results 48

4.3 VOC and the Amazon Elastic Compute Cloud(EC2) 49

4.3.1 Experiments 49

4.3.2 Results 54

5 Conclusions 55

Bibliography 57

List of Tables

4.1 Observed makespan lengths and system throughputs for 10 overlay experiment trials 48

List of Figures

1.1 General Rep. of Grids . . . 2
1.2 Serial vs. Parallel Computation . . . 2
1.3 Distributed computing . . . 5
1.4 Cloud Computing . . . 7
1.5 Condor Architecture . . . 9
1.6 Client-Server Architecture . . . 11
1.7 Peer-to-Peer System . . . 12
1.8 Distributed Hash Table . . . 13
1.9 DHT Implementation . . . 14
1.10 IPOP Architecture . . . 16
1.11 Structure of IPOP . . . 17
1.12 Virtual Organization Clusters . . . 21

3.1 IPOP as a STUN server . . . 32
3.2 IPOP DHCP Configuration File . . . 34
3.4 IPOP Bootstrap Configuration File . . . 35
3.3 IPOP Namespace Configuration File . . . 35
3.5 IPOP Node Configuration File . . . 36
3.6 Single Site VOC . . . 40
3.7 VOC over a single site . . . 41
3.8 Multiple Grid Site VOC . . . 42

4.1 Short Jobs without the Overlay Network . . . 47
4.2 Short Jobs with the Overlay Network . . . 47
4.3 Long Jobs without the Overlay Network . . . 47
4.4 Long Jobs with the Overlay Network . . . 48
4.5 Amazon EC2 Primary Distributed Architecture[2] . . . 49
4.6 Amazon Home UI . . . 50
4.7 Amazon Instance UI . . . 50
4.8 Short Jobs executing in Multiple Grid sites over IPOP . . . 52
4.9 Long Jobs executing in Multiple Grid sites over IPOP . . . 53

Listings

Chapter 1

Introduction

"*Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.*" - Ian Foster [41]. This statement, which was made by Ian Foster in 2001, serves as a standard for all research and development conducted in the grid community. The key difference between Grid computing and Distributed computing is in the allocation of compute resources. Grid computing requires access to compute resources that are exclusively dedicated which includes powerful workstations, super computers and servers. Distributed computing, on the other hand, does not have such a stringent requirement. It requires only a subset of resources from any single computation unit. This means that at any instant of time, a system could be 'live' (actively deployed in a distributed environment) and be used by the owner of the system simultaneously. Such operations are abstracted away from the user in such a way that his work is never affected. Despite these differences, the scientific community has agreed on the consensus that Grid computing is a subset of Distributed computing.

Large applications are often required for scientific simulations, research, modeling and other applications. Single computation units, despite having multiple processors, huge amounts of RAM and disk space, can neither satisfy the computing requirement of ever growing data sets nor provide results in a reasonable amount of time. Therefore the timing for a gradual but definite adaptation of large-scale distributed systems is indeed warranted and befitting[39].

Clemson University has a large supercomputing infrastructure, in the Palmetto Cluster. It has over 6000 cores and is ranked 89 among the TOP 500 Supercomputing Sites in the World.

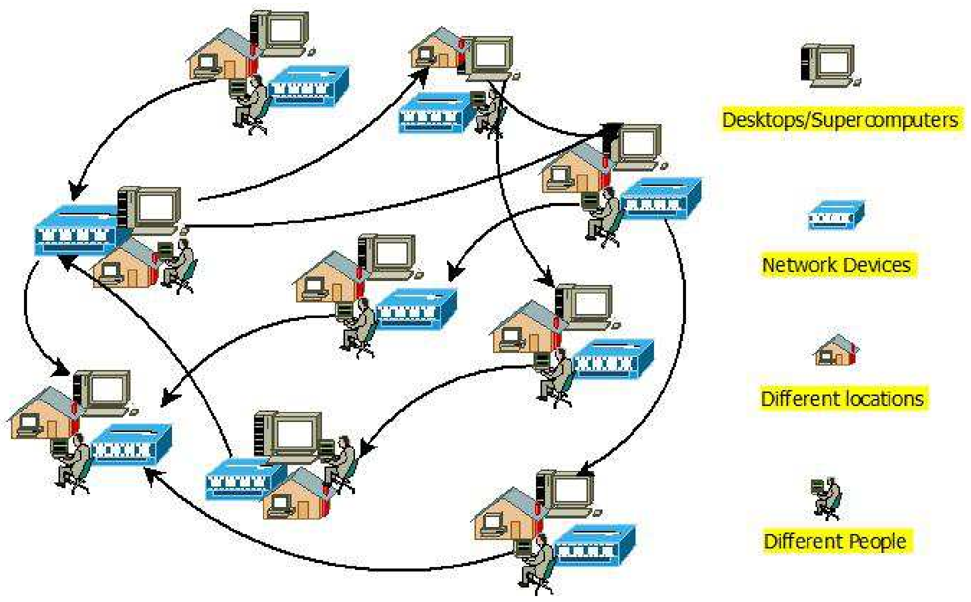


Figure 1.1: A General Representation of Grids comprising not just Computing and Networking hardware but geographic locations and people

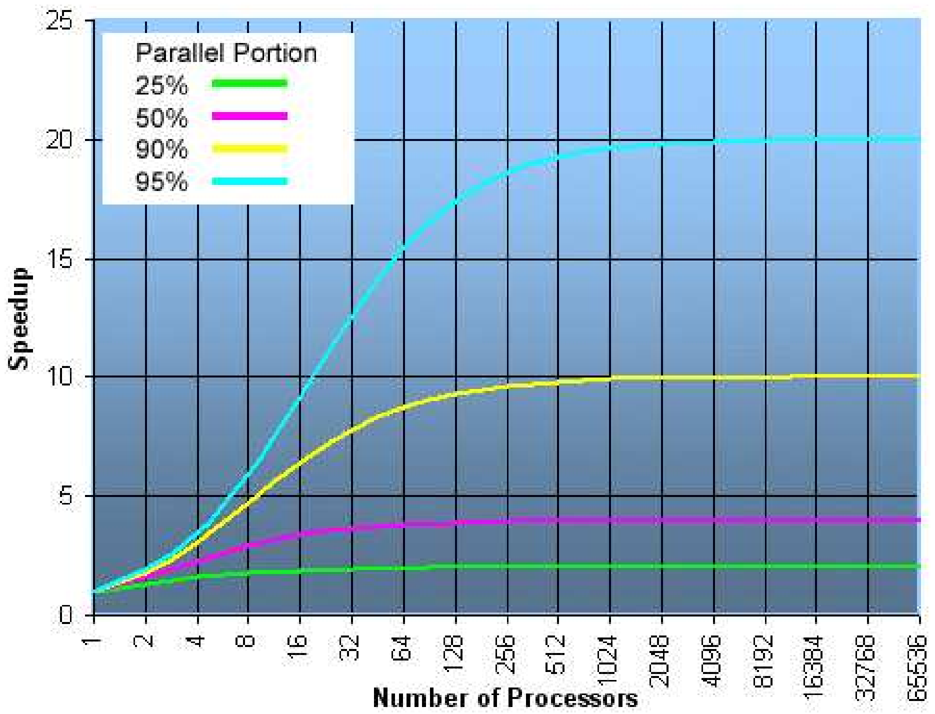


Figure 1.2: Serial vs. Parallel Computation - The effect of parallelizing code and using distributed resources[10].

The Cyber Infrastructure Research Group[31] at Clemson is the leading developer and strategist for distributed computing and infrastructure on campus. Led by Dr. Sebastien Goasguen, the group looks to leverage campus wide resources including the Palmetto Cluster to research and develop distributed applications[3].

The author describes virtualization, which is yet another important concept in distributed computing and makes up a significant part when defining the term Cloud Computing. These are discussed in detail in Sections1.4, 2 and 3. This thesis will focus on the networking infrastructure that was researched and deployed as part of the Virtual Organizations Cluster model[53]. A large-scale distributed environment with systems deployed in widely separated geographic locations needs to have a communication system as one of the foundations. But there are many hurdles to cross before such a networking environment can be provided. Some of these include site administrative policies, NATs, firewalls and various other system wide constraints that arise from the management of heterogeneous systems. The predominant networking direction has been toward a client-server architecture. The author takes a new direction - the Peer-to-peer approach, the advantages and details of which will be discussed in detail in the forthcoming sections of this thesis.

1.1 Research Questions

In an effort to develop an efficient and usable distributed infrastructure, this thesis looks to answer the following research question:

1. *"Given a Virtual Organization Cluster, is it possible to resize based on work load?"*
2. *"Is it possible to acquire off-site grid resources if the local resources are exhausted? If so, how can this be done?"*
3. *"What impact would the acquisition of these resources have on site-administrators and end-users?"*

1.2 Motivation for Research

The primary motivation that steered this research was to develop a system that:

1. Enabled Scientists(and other users) to dynamically acquire large amounts of computer resources that was cost-efficient and,
2. Requires less administration and maintenance.
3. Requires little-to-no adaptability from the user both in terms of skills and software.
4. Increase system utilization (whether in a purely grid environment or purely distributed environment(clouds))

1.3 Distributed Computing

"A Distributed System is an application that executes a collection of protocols to coordinate the actions of multiple processes on a network, such that all components cooperate together to perform a single or small set of related tasks"[13]. A quintessential example for such a system is the Condor - High Throughput Computing Job Scheduler[64]. An introduction to Condor is detailed in Section1.5 and a description of how it was efficiently deployed with the Clemson VOC model is detailed in Sections 1.9 and 3.

As scientific data sets and research requirements increase rapidly, existing infrastructure including high-end servers suffer increase CPU usage and often crash. It is necessary to have sufficient data and resources backed up so that important systems which include web servers and hospital emergency systems can function efficiently at another location(geographically or virtually). Parallel/Distributed systems are used:

- To Divide and Conquer. These systems are used to run applications that are designed to run either concurrently or exhibit some level of parallelism. Such applications require parallelism exclusively, as in the case of simulators and modeling systems, or may require it for the sole purpose of running single processor applications within a much smaller time frame.
- In Data Redundancy and Replication: As mentioned above, web servers and other essential systems need to be up-and-running all the time. Therefore, server load needs to often be split between multiple instances of the same system. This will reduce the load on one single server in a seamless manner that abstracts all high level implementation detail from the end-user. Load-balancing between servers(systems) on the same LAN and WAN is one use case of a distributed system.

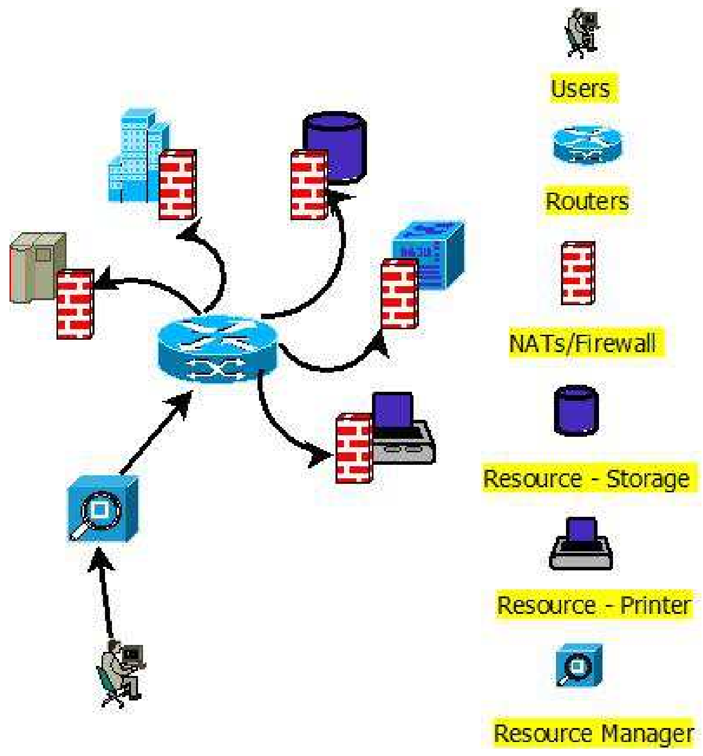


Figure 1.3: A General view of Distributed Computing with focus on resource managing, authentication and sharing

1.4 Virtualization and Cloud Computing

An environment that 'simulates' or 'models' a set of features that are independent of the physical environment is called a Virtual Environment. The term 'Virtual Machine' originated from the IBM M44/44X project in the 1960's[32]. The IBM M44/44X was an experimental computer system, designed and operated at IBM's Thomas J. Watson Research Center at Yorktown Heights, New York. It was based on an IBM 7044 (the 'M44'), and simulated multiple 7044 virtual machines (the '44X'), using both hardware and software. Key team members were Dave Sayre and Rob Nelson. This was a groundbreaking machine, used to explore paging, the virtual machine concept, and computer performance measurement. It was purely a research system, and was cited in 1981 by Peter Denning as an outstanding example of experimental computer science[12].

Popek and Goldberg[58] introduced a classification of instructions of ISA, that were necessary to support Virtualization, into 3 different groups:

1. Privileged instructions - Those that trap if the processor is in user mode and do not trap if it

is in system mode.

2. Control sensitive instructions - Those that attempt to change the configuration of resources in the system.

3. Behavior sensitive instructions - Those whose behavior or result depends on the configuration of resources (the content of the relocation register or the processor's mode).

The main result of Popek and Goldberg's analysis can then be expressed as follows.

- Theorem 1. For any conventional third-generation computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions. Intuitively, the theorem states that to build a VMM it is sufficient that all instructions that could affect the correct functioning of the VMM (sensitive instructions) always trap and pass control to the VMM. This guarantees the resource control property. Non-privileged instructions must instead be executed natively (i.e., efficiently). The holding of the equivalence property also follows.

- Theorem 2. A conventional third-generation computer is recursively virtualizable if a) it is virtualizable and b) a VMM without any timing dependencies can be constructed for it.

Virtual machines are separated into two major categories, based on their use and degree of correspondence to any real machine. A system virtual machine provides a complete system platform which supports the execution of a complete operating system (OS). In contrast, a process virtual machine is designed to run a single program, which means that it supports a single process. An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine—it cannot break out of its virtual world.

Now that we have a good understanding of how virtualization works, we can delve into the realm of Cloud Computing. Cloud computing is an architecture; a new technology; a new perception. Cloud Computing is accessing and providing compute resources via the Internet or a private network. That means storage, computing, entertainment hosting, etc., can be administered remotely without even having the necessary infrastructure. Among the plethora of Internet definition's, there are some key attributes that characterize 'Clouds'[5]:

- Dynamic computing infrastructure

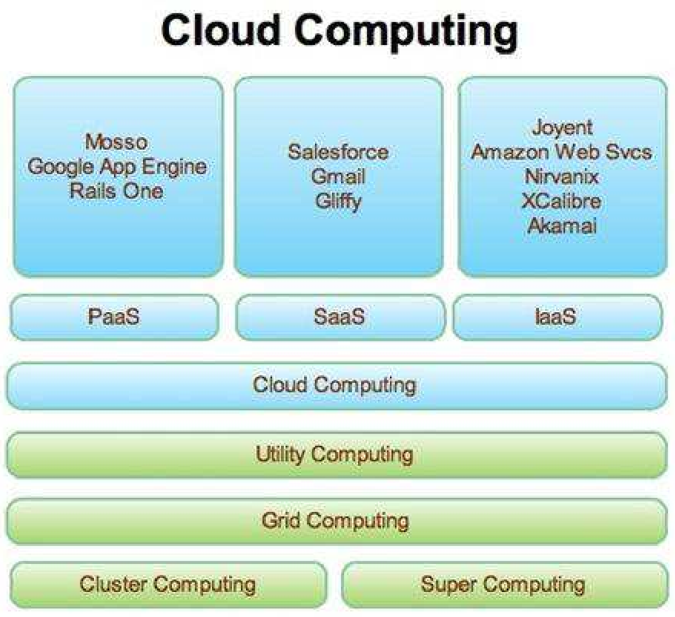


Figure 1.4: Cloud Computing - A pictorial representation[4]

- IT service-centric approach
- Self-service based usage model
- Minimally or Self managed platform
- Consumption based billing

A technical definition is "a computing capability that provides an abstraction between the computing resource and its underlying technical architecture (e.g., servers, storage, networks), enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction."[6] This definition states that clouds have five essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Narrowly speaking, cloud computing is client-server computing that abstract the details of the server away – one requests a service (resource), not a specific server (machine). However, cloud computing may be conflated with other terms, including client-server and utility computing, and the term has been criticized as vague and referring to "everything that we currently do"[6].

1.5 High Throughput Computing and Condor

For many experimental scientists, scientific progress and quality of research are strongly linked to computing throughput. In other words, most scientists are concerned with how many floating point operations per month or per year they can extract from their computing environment rather than the number of such operations the environment can provide them per second or minute. Floating point operations per second (FLOPS) has been the yardstick used by most High Performance Computing (HPC) efforts to evaluate their systems. Little attention has been devoted by the computing community to environments that can deliver large amounts of processing capacity over long periods of time. We refer to such environments as High Throughput Computing (HTC) environments[21].

The key to HTC is effective management and utilization of all available computing resources. Since the computing needs of most scientists can be satisfied these days by commodity CPUs and memory, high efficiency is not playing a major role in a HTC environment. The main challenge a typical HTC environment faces is how to maximize the amount of resources accessible to its customers. Distributed ownership of computing resources is the major obstacle such an environment has to overcome in order to expand the pool of resources it can draw from. Recent trends in the cost/performance ratio of computer hardware have placed the control (ownership) over powerful computing resources in the hands of individuals and small groups. These distributed owners will be willing to include their resources in a HTC environment only after they are convinced that their needs will be addressed and their rights protected[21].

Condor is an open source high-throughput computing software framework for coarse-grained distributed parallelization of computationally intensive tasks. It can be used to manage workload on a dedicated cluster of computers, and/or to farm out work to idle desktop computers — so-called cycle scavenging. Condor runs on Linux, Unix, Mac OS X, FreeBSD, and contemporary Windows operating systems. Condor can seamlessly integrate both dedicated resources (rack-mounted clusters) and non-dedicated desktop machines (cycle scavenging) into one computing environment.

By way of example, the NASA Advanced Supercomputing facility (NAS) Condor pool consists of approximately 350 SGI and Sun workstations purchased and used for software development, visualization, email, document preparation, etc. Each workstation runs a daemon that watches user I/O and CPU load. When a workstation has been idle for two hours, a job from the batch queue

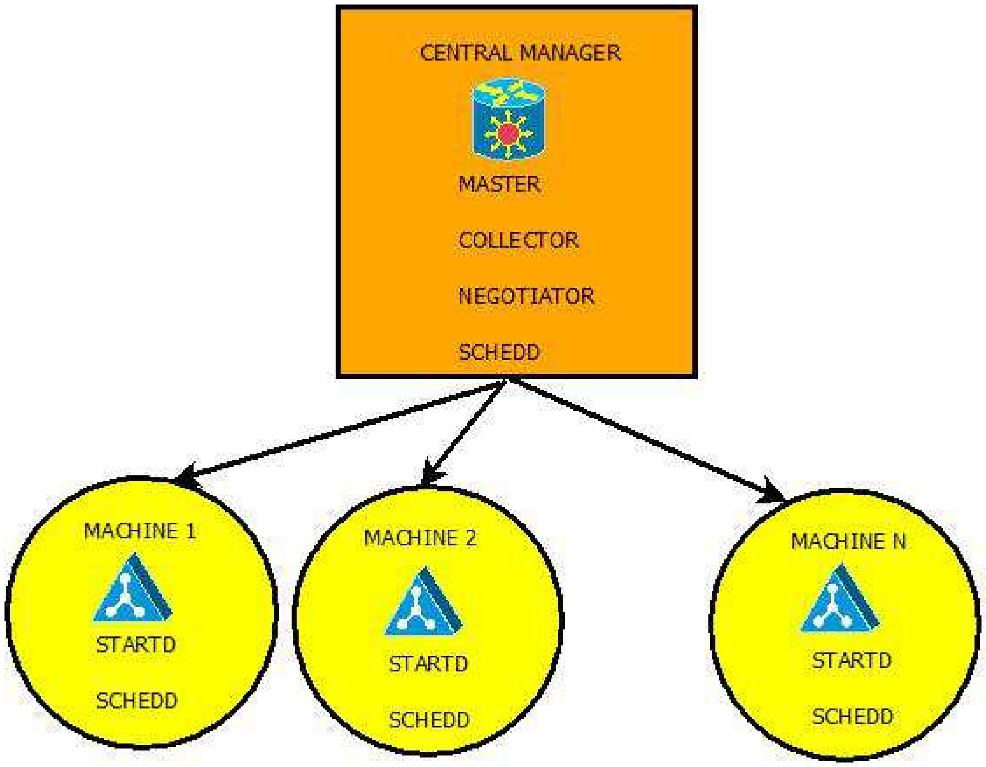


Figure 1.5: The Condor Architecture. It is built around the central manager which schedules and maintains jobs among the various client/submit machines.

is assigned to the workstation and will run until the daemon detects a keystroke, mouse motion, or high non-Condor CPU usage. At that point, the job will be removed from the workstation and placed back on the batch queue.

Condor can run both sequential and parallel jobs. Sequential jobs can be run in several different "universes", including "vanilla" which provides the ability to run most "batch ready" programs, and "standard universe" in which the target application is re-linked with the Condor I/O library which provides for remote job I/O and job checkpointing. Condor also provides a "local universe" which allows jobs to run on the "submit host".[34]

1.5.1 Condor at Clemson

Condor has been successfully deployed in approximately 3000 machines in over 50 locations around the Clemson University campus. Microsoft Windows and Linux are the primary operating systems that are deployed on Campus. Clemson has deployed a Condor pool consisting of Windows Vista machines in the public computer labs and several other groups of machines (Linux, Solaris, etc.). These machines are available to Clemson faculty, students, and staff with high-throughput computing needs. Users can create their own Condor submit machines by downloading the appropriate software, and can even contribute their own idle cycles to the pool

Beside these machines, the Palmetto Cluster[16] houses over 1000 high end servers that are a capable of providing 69 TFlops making it among the top 100 of the fastest supercomputers in the world.

1.6 Peer to Peer Architecture vs. Client-Server Architecture

In a Client-Server architecture, there is one (or more) dedicated machine(s) that function as file-servers, storage-machines or print servers. Several client machines require one or more of the services that these servers provide. The networking in such an environment comprises primarily of a dedicated connection between the server and its clients. Communication between clients is facilitated through the server. One advantage of such a system is the ease in setting up a connection. Security and routing also factor in on why so many systems are based on this architecture. In a client-server environment like Windows NT or Novell NetWare, files are stored on a centralized, high speed file server PC that is made available to client PCs. Network access speeds are usually faster than those

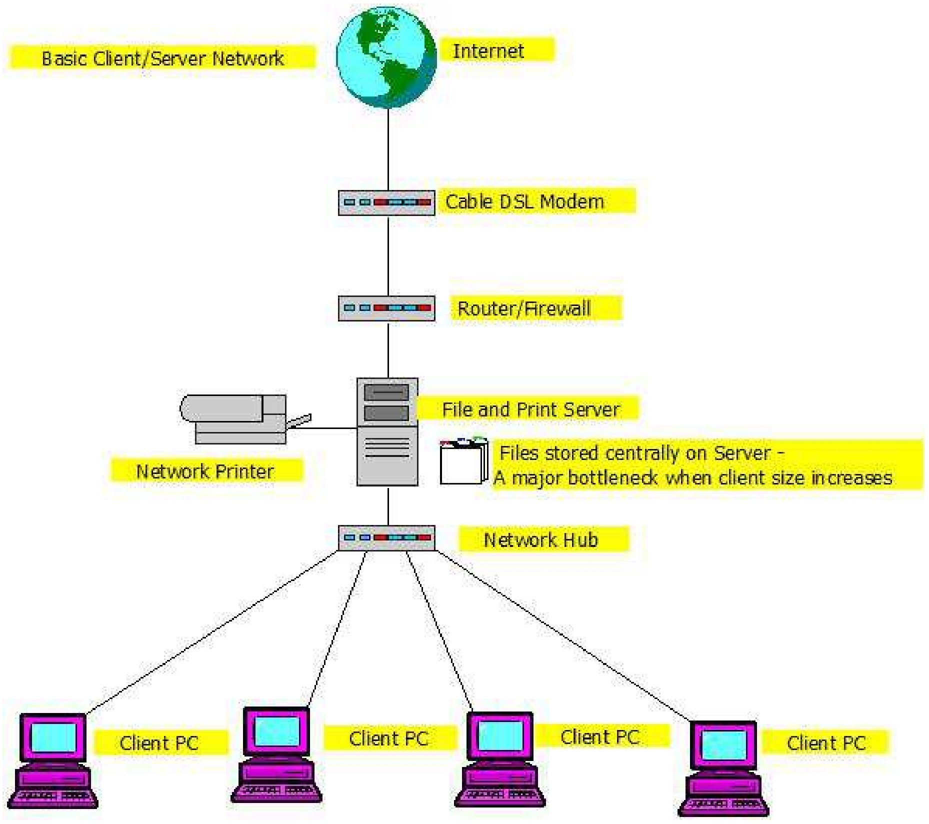


Figure 1.6: A very basic Client-Server Architecture

found on peer-to-peer networks, which is reasonable given the vast numbers of clients that this architecture can support. Nearly all network services like printing and electronic mail are routed through the file server, which allows networking tasks to be tracked. Inefficient network segments can be reworked to make them faster, and user's activities can be closely monitored. Public data and applications are stored on the file server, where they are run from client PCs locations, which makes upgrading software a simple task–network administrators can simply upgrade the applications stored on the file server, rather than having to physically upgrade each client PC[8].

In a peer-to-peer system, there are no single servers or dedicated machines. Each machine functions in a dual client-server mode: functioning as a client requiring service and functioning as a server providing service - at the same time. A peer-to-peer network allows two or more PCs to pool their resources together. Individual resources like disk drives, CD-ROM drives, and even printers are transformed into shared, collective resources that are accessible from every PC.

Unlike client-server networks, where network information is stored on a centralized file server

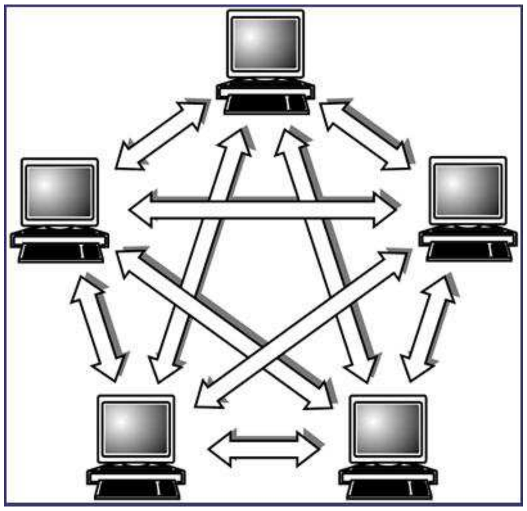


Figure 1.7: A Peer-to-Peer system has no dedicated servers. Instead each machine in the system acts as a client and server. This ensures high availability and fault tolerance

PC and made available to tens, hundreds, or thousands client PCs, the information stored across peer-to-peer networks is uniquely decentralized. Because peer-to-peer PCs have their own hard disk drives that are accessible by all computers, each PC acts as both a client (information requestor) and a server (information provider).

As an example, a peer-to-peer network can be built with either 10BaseT cabling and a hub or with a thin coax backbone. 10BaseT is best for small workgroups of 16 or fewer users that do not span long distances, or for workgroups that have one or more portable computers that may be disconnected from the network from time to time. After the networking hardware has been installed, a peer-to-peer network software package must be installed onto all of the PCs. Such a package allows information to be transferred back and forth between the PCs, hard disks, and other devices when users request it[8].

1.7 Distributed Hash Tables

In the world of decentralization, distributed hash tables (DHTs) have had a revolutionary effect. The chaotic, ad hoc topologies of the first-generation peer-to-peer architectures have been superseded by a set of topologies with emergent order, provable properties and excellent performance. Knowledge and understanding of DHT algorithms is going to be a key ingredient in future developments of distributed applications.

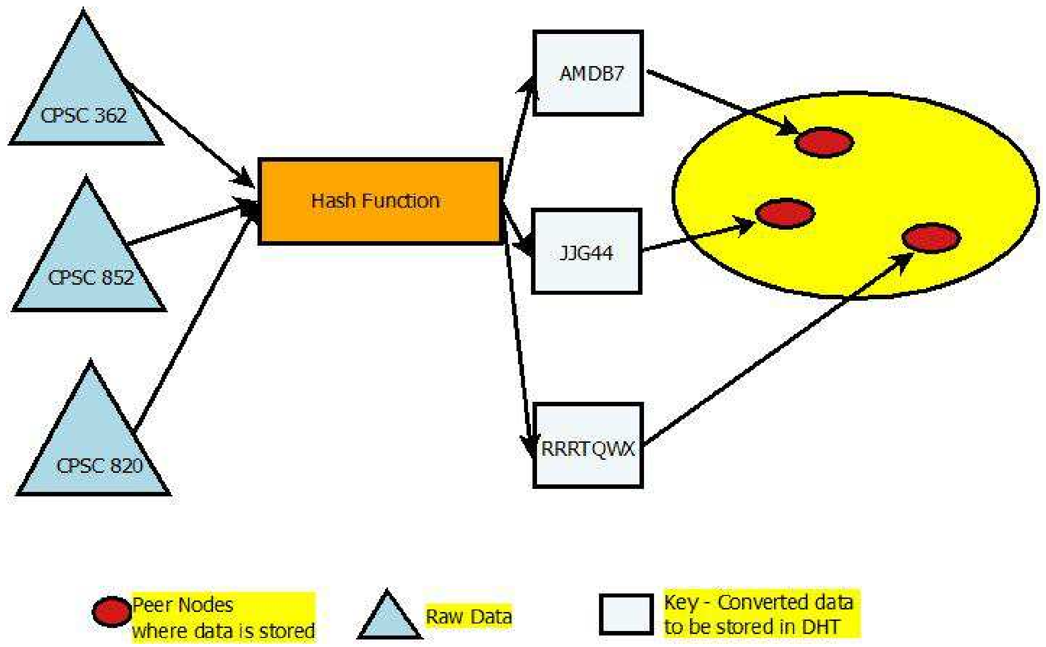

CPSC 362
CPSC 852
CPSC 820
Hash Function
AMDB7
JJG44
RRRTQWX
Peer Nodes where data is stored
Raw Data
Key - Converted data to be stored in DHT

Figure 1.8: The Basic functioning of most Distributed Hash Tables, where human readable data is converted into a secure key using a hash function and then distributed across a known set of machines.

A number of research DHTs have been developed by universities, picked up by the Open Source community and implemented. A few of them are discussed in 2, including Brunet[26] which is a DHT library that IPOP[26] is based on. A few proprietary implementations exist as well, but currently none are available as SDKs; rather, they are embedded in commercially available products. Each DHT scheme generally is pitched as being an entity unto itself, different from all other schemes. In actuality, the various available schemes all fit into a multidimensional matrix. Take one, make a few tweaks and you end up with one of the other ones. Existing research DHTs, such as Chord, Kademlia and Pastry, therefore are starting points for the development of your own custom schemes. Each has properties that can be combined in a multitude of ways. In order to fully express the spectrum of options, let's start with a basic design and then add complexity in order to gain useful properties[9].

Basically, a DHT performs the functions of a hash table. You can store a key and value pair, and you can look up a value if you have the key. Values are not necessarily persisted on disk, although you certainly could base a DHT on top of a persistent hash table, such as Berkeley DB; and in fact, this has been done. The interesting thing about DHTs is that storage and lookups are distributed

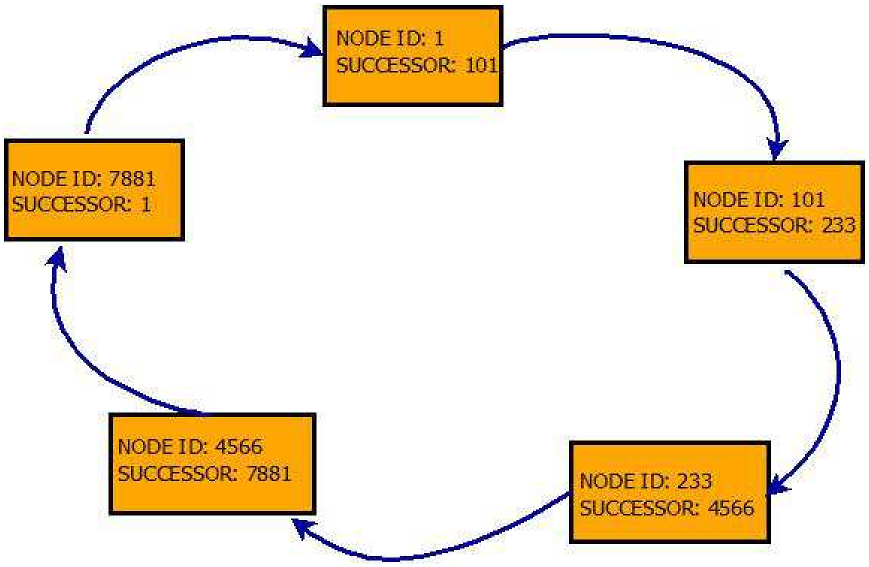


Figure 1.9: In a real DHT implementation, each node would be running on a different machine and all calls to them would be needed to be communicated over some socket protocol. Each node is itself a standard hash table. All we need to do, to store or retrieve a value from the hash table is find the appropriate node in the network, then store it in that nodes hash table. Each node points to another node called its successor. The last node points to the first node and forming the network overlay.

among multiple machines. Unlike existing master/slave database replication architectures, all nodes are peers that can join and leave the network freely. Despite the apparent chaos of periodic random changes to the membership of the network, DHTs make provable guarantees about performance.[9]

The structure of a DHT can be decomposed into several main components. The foundation is an abstract keyspace, such as the set of 160-bit strings. A keyspace partitioning scheme splits ownership of this keyspace among the participating nodes. An overlay network then connects the nodes, allowing them to find the owner of any given key in the keyspace.

DHTs characteristically emphasize the following properties[7]:

- Decentralization: the nodes collectively form the system without any central coordination.
- Scalability: the system should function efficiently even with thousands or millions of nodes.
- Fault tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.

A key technique used to achieve these goals is that any one node needs to coordinate with only a few other nodes in the system – most commonly, O(log n) of the n participants (see below)

– so that only a limited amount of work needs to be done for each change in membership.

Some DHT designs seek to be secure against malicious participants and to allow participants to remain anonymous, though this is less common than in many other peer-to-peer (especially file sharing) systems; see anonymous P2P.

Finally, DHTs must deal with more traditional distributed systems issues such as load balancing, data integrity, and performance (in particular, ensuring that operations such as routing and data storage or retrieval complete quickly)[7].

Some common applications where DHTs are significantly used include:

- File sharing: store the hash of the file in the DHT, and hashes of keywords relevant to the file. People looking for the file can then search by keyword, or if they know the exact hash of the file, search for that hash.
- Finding people (e.g. in a chat program): store the hash of their public key, and hashes of their names to the DHT.
- Providing a service (e.g. distributed backup): store a particular hash to the DHT in order to indicate that you are willing to provide a particular service (perhaps the hash of a short description of that service).

1.8 From P2P to DHT to Brunet, then came - IP Over P2P (IPOP)

Designed at the University of Florida at Gainseville, IPOP(IP-over-P2P) is a user-level overlay networking system that supports IP tunneling on UDP and TCP transports, including support for NAT traversal for UDP using hole-punching.

P2P networks can be made self-configurable. They allow user mobility and are scalable. They also provide extremely robust service and hence motivate the choice of P2P routing as the basis for IPOP. Through IPOP, resources spanning multiple domains can be aggregated into a virtual IP network providing bidirectional connectivity[42]. The protocols support seamless, self-configured addition of nodes to a virtual IP network and might be classified as a P2P protocol for VPN (virtual private networks). The IPOP virtual IP address space is routable within the P2P overlay, however it is decoupled from the address space of the physical Internet infrastructure — IP packets are

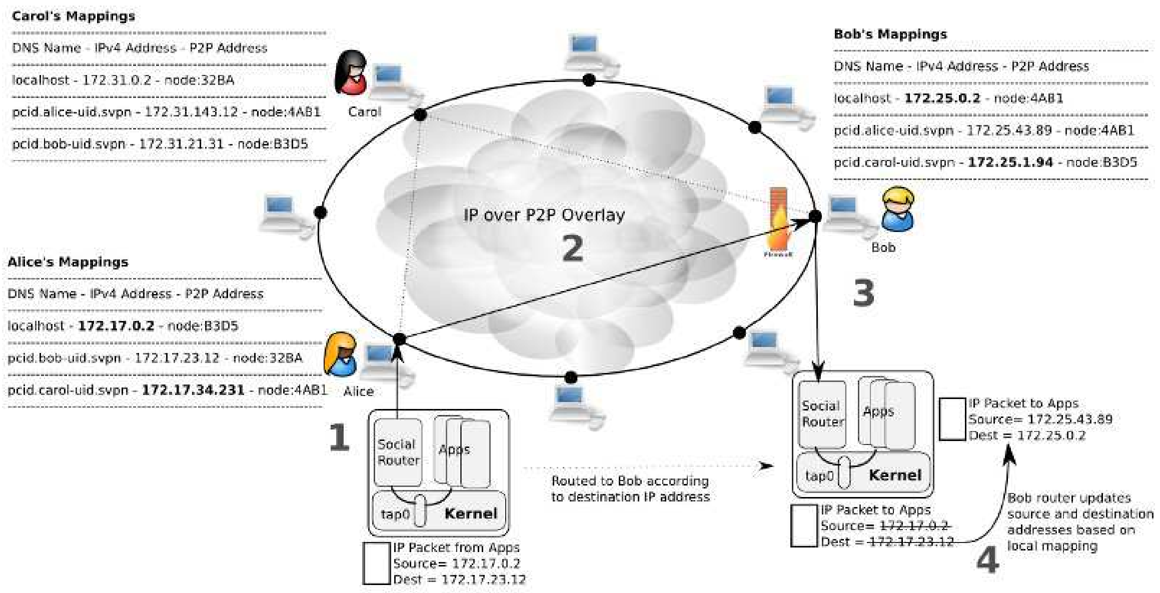

Carol's Mappings
DNS Name - IPv4 Address - P2P Address
localhost - 172.31.0.2 - node:32BA
pcid.alice-uid.svpn - 172.31.143.12 - node:4AB1
pcid.bob-uid.svpn - 172.31.21.31 - node:B3D5
Carol
Bob's Mappings
DNS Name - IPv4 Address - P2P Address
localhost - 172.25.0.2 - node:4AB1
pcid.alice-uid.svpn - 172.25.43.89 - node:4AB1
pcid.carol-uid.svpn - 172.25.1.94 - node:B3D5
IP over P2P Overlay
2
Bob
3
Alice's Mappings
DNS Name - IPv4 Address - P2P Address
localhost - 172.17.0.2 - node:B3D5
pcid.bob-uid.svpn - 172.17.23.12 - node:32BA
pcid.carol-uid.svpn - 172.17.34.231 - node:4AB1
Alice
1
Social Router
Apps
tap0
Kernel
IP Packet from Apps
Source= 172.17.0.2
Dest = 172.17.23.12
Routed to Bob according to destination IP address
Social Router
Apps
tap0
Kernel
IP Packet to Apps
Source= 172.25.43.89
Dest = 172.25.0.2
IP Packet to Apps
Source= ~~172.17.0.2~~
Dest = ~~172.17.23.12~~
4
Bob router updates source and destination addresses based on local mapping

Figure 1.10: IPOP Architecture[15]

payloads that tunnel through the P2P overlay. Current network virtualization techniques for Grid computing require an administrator to setup overlay routing tables. Hence, the process of adding, configuring and managing clients and servers that route traffic within the overlay is difficult to scale. Although topology adaptation is possible using techniques proposed, adaptive routes are coordinated by a centralized server. Both VNET[27] and VIOLIN[47] can provide a robust overlay through redundancy. However, the effort required to preserve robustness would increase every time a new node is added and the network grows in size. In contrast, the use of P2P routing to overlay virtual IP traffic differentiates IPOP from existing network virtualization solutions with respect to the following issues[66]:

- Scalability: Network management in a P2P-based routing overlay scales to large numbers because routing information is naturally self-configured, decentralized, and dynamically adaptive in response to nodes joining and leaving. Adding a new resource to the virtual IP network requires minimal effort, which is independent of current size of the network. Performance scaling leverages from the fact that each node contributes bandwidth and processing power so that the resources of a system grow as nodes join.

- Resiliency: P2P networks are robust even in the face of high failure rates. An IP-over-P2P

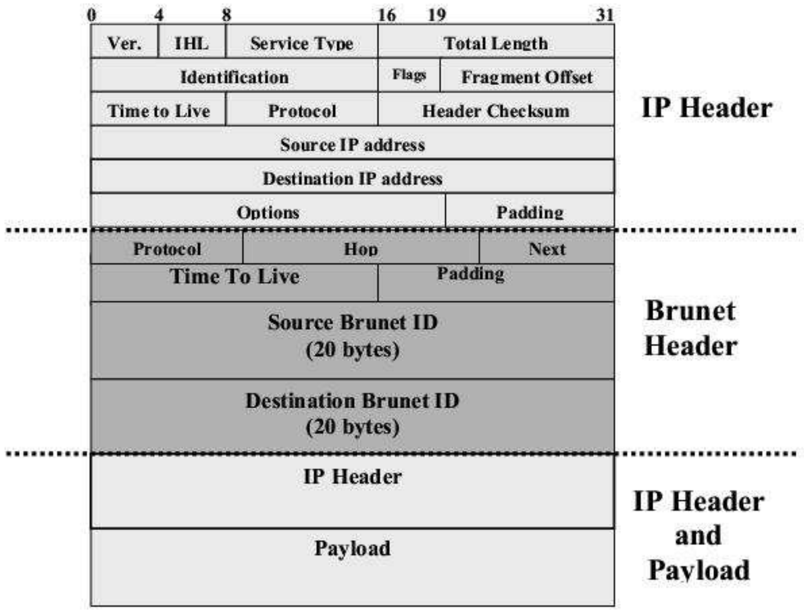


Figure 1.11: Structure of an IPOP encapsulated IP Packet[42]

overlay benefits from the synergy of fault-tolerant techniques applied at different levels. The IPOP overlay dynamically adapts routing of IP packets as nodes fail or leave the network; even if packets are dropped by such nodes, IP and other protocols above it in the network stack have been designed to cope with such transient failures. The use of P2P overlays to provide robust routing has also been described.

- Accessibility: Existing network virtualization techniques can leverage mechanisms described to cross NAT/Firewalls. These approaches require setting up globally reachable STUN or STUNT servers that aid building the necessary NAT state by carefully crafted exchange of packets. With P2P networks, each overlay node can provide this functionality for detection of NATs and their subsequent traversal. This approach is decentralized and introduces no dedicated servers[42].

1.8.1 Architectural Overview

The IPOP architecture has two key components: a virtualized network interface for capturing and injecting IP packets into the virtual network, and a P2P routing substrate that encapsulates, tunnels and routes packets within the overlay. Let us examine how IP packets are captured from and injected to the host, and then how IP packets are routed on the Brunet P2P network[66][44][43].

- **Capturing IP from the Host** - The prototype captures packets from a virtual device attached to the source host, tunnels them over the application-level Brunet P2P overlay, and then injects the packet back into a virtual device on the destination host. IPOP uses the tap device, which appears as a virtual network interface inside the host, and is available for several Unix platforms and Windows. Through tap, Ethernet frames injected by the the kernel on the virtual interface can be read by a user-level application, and vice versa. IPOP runs as a C# program over the Brunet library. It reads and writes Ethernet frames from the tap device, and uses Brunet to assemble IP packets inside P2P packets and to route them to their destination. While IPOP sees Ethernet frames, it only routes IP packets; non-IP traffic, notably ARP traffic, is contained within the host. A host running IPOP is a P2P node, and can act as data source, receiver and router all at the same time. Since we are dealing with Ethernet level frames at the IPOP hosts, we may come across non-IP based frames such as ARP and RARP. The nonexistent gateway has a unique IP address in the virtual address space, and its Ethernet address is configured to not conflict with that of the local tap device. As a result, it is possible for the host to only send out the IP- based Ethernet frames and contain ARP requests locally.

- **Routing IP on the P2P Overlay** - When a host needs to be added to the IPOP virtual network, the host administrator is required to set up a tap device in the host, and bring it up with an IP address that is unique in the virtual IP address space. IPOP runs on the host as a P2P node whose address is the 160 bit SHA-1 hash of the IP address of the tap device. The application running on one host generates IP-based traffic which the kernel transmits through the 'tap0' interface. The Ethernet frame is captured, an IP packet extracted from it, and then encapsulated inside a P2P packet. The P2P packet, in turn, is sent to the P2P node whose address is the 160 bit SHA-1 hash of the destination IP address. Once the P2P packet is received at the destination node, the IP packet is extracted, an Ethernet frame is built from it, and the frame is written to the tap0 interface on the host. IPOP has been designed with the goal of providing open access among a set of trusted nodes spanning multiple sites through network virtualization.

- **IPOP extensions for virtual machines -** A driving application for IPOP in the context of Grid computing is to interconnect O/S virtual machines (e.g. Xen[25], KVM[60]) with the IP overlay, thereby virtualizing key Grid resources. A machine (physical or virtual) running IPOP must have a way of connecting to other P2P nodes, without necessarily having a public IP address. We have implemented and tested with Virtual Box's, KVM's amd Xen's bridged and NAT network interfaces. In all cases IPOP runs within the VM. In this case, the VM's virtual Ethernet card is configured with an IP address in the virtual address space, and routes and static ARP entries are set as described earlier. The virtual Ethernet card of the UML guest attaches itself to a tap device on the host, and the UML kernel uses tap reads and writes to transmit Ethernet frames. A tap device can be opened by at most one process, and hence IPOP cannot directly read and write Ethernet frames from the same tap device. The tap device is therefore bridged [4] to another tap device on the host, from which IPOP can read and write Ethernet frames.

- **Crossing firewalls and NATs -** One of the many benefits we derive from overlaying IP on a P2P network is the overlay we use, Brunet[26], already deals with connecting nodes which are behind firewalls and NAT devices. Of the four types of NATs used today, all have the property that if a UDP packet is sent from IP address A port to IP address B port, the NAT device will allow packets from IP address B port to flow to IP address A port. Any system that does not permit this is broken because it is allowing outbound packets without allowing any response to those packets, which meets the requirements of almost no applications. In addition to the above property, three out of four of the common NAT types (all but the symmetric) use the same mapping for the NAT's port internal (IP, port) pair. Thus each connection in the P2P network is an opportunity for a node to discover if any IP translation is going on, and if so record its translated address. Once a node discovers that its address is being translated, it advertises that translated address to another nodes over the Brunet network. Furthermore, this approach is decentralized and introduces no single points of failure or dedicated servers (unlike the STUN protocol). While this NAT/firewall traversal may seem like a subversion of network policies, this is arguably not the case. In fact, both nodes in this scenario are actively attempting to contact the other, and as such, any unrequested packets are still filtered by the NAT or firewall.

- **Multiple IPs and mobility** The current solution of mapping IP addresses to Brunet addresses using SHA-1 hashes requires one P2P node per IP address. Because of this requirement, a single IPOP node running on a host cannot ”route for” multiple virtual IPs (e.g.. multiple VMs hosted by a physical machine). The problem is aggravated when virtual IP addresses are mobile — a situation that can occur when virtual machines are allowed to migrate[47][67]. A solution to this problem involves using the P2P system as a distributed hash table (DHT). This node is called “Brunet- ARP-Mapper”. Now when a node has an IP packet to send, it inquires about the brunet destination from the corresponding “Brunet-ARP-Mapper” whose Brunet address is the 160 bit SHA-1 hash of the destination IP address. This information can then be cached at the source node. When a VM migrates (retaining its IP address), the information is updated at its corresponding “Brunet-ARP-Mapper”.

1.9 Virtual Organization Cluster (VOC)

The Virtual Organization Cluster Model was developed by Michael Murphy and Sebastien Goasguen at Clemson University. Parts of this Section have been reproduced from previous work found in Publication[53].Virtual Organizations (VOs) enable scientists to collaborate using diverse, geographically distributed computing resources. Requirements and membership of these VOs often change dynamically, with objectives and computing resource needs changing over time. Given the diverse nature of VOs, as well as the challenges involved in providing suitable computing environments to each VO, Virtual Machines (VMs) are a promising abstraction mechanism for providing grid computing services. Such VMs may be migrated from system to system to effect load balancing and system maintenance tasks. Cluster computing systems, including services and middleware that can take advantage of several available hypervisors, have already been constructed inside VMs. However, a cohesive view of virtual clusters for grid-based VOs has not been presented to date[53].

Implementing computational clusters with traditional multiprogramming systems may result in complex systems that require different software sets for different users. Each user is also limited to the software selection chosen by a single system administrative entity, which may be different from the organization sponsoring the user. Virtualization provides a mechanism by which each user entity might be given its own computational environment. Such virtualized environments would permit greater end-user customization at the expense of some computational overhead. The primary motivation for the work described here is to enable a scalable, easy-to-maintain system on which each

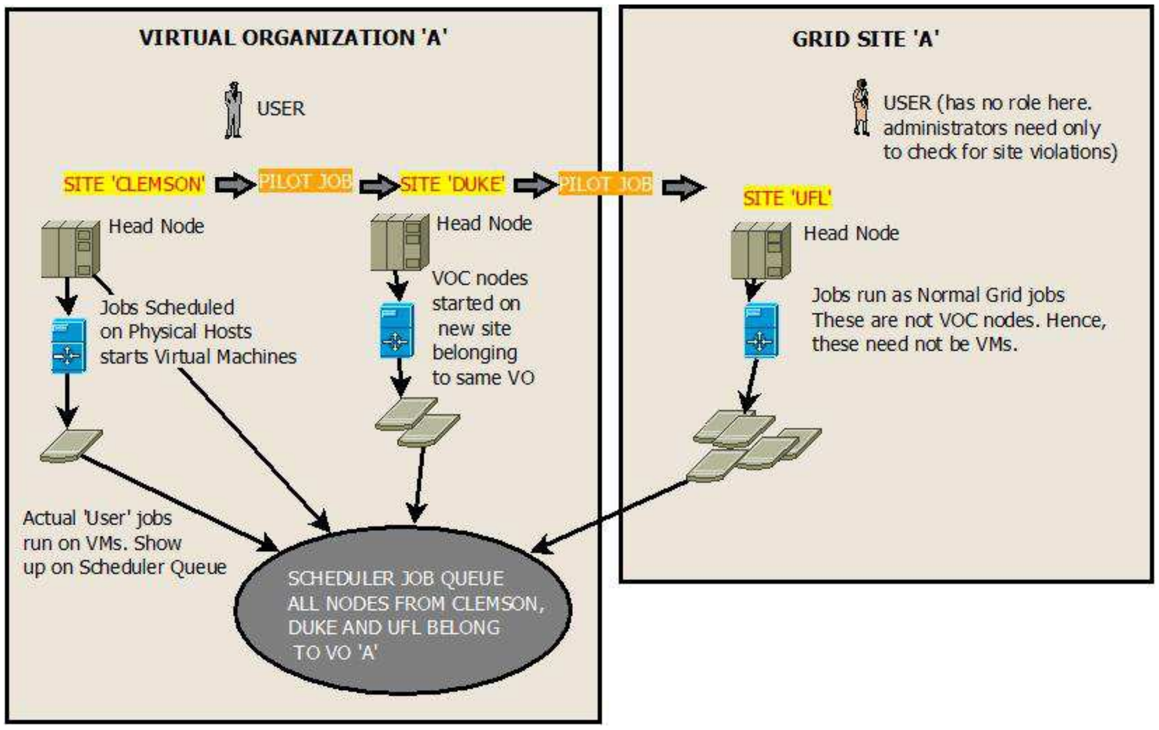

VIRTUAL ORGANIZATION 'A'
USER
SITE 'CLEMSON'
PILOT JOB
SITE 'DUKE'
PILOT JOB
Head Node
Jobs Scheduled on Physical Hosts starts Virtual Machines
Head Node
VOC nodes started on new site belonging to same VO
Actual 'User' jobs run on VMs. Show up on Scheduler Queue
SCHEDULER JOB QUEUE ALL NODES FROM CLEMSON, DUKE AND UFL BELONG TO VO 'A'
GRID SITE 'A'
USER (has no role here. administrators need only to check for site violations)
SITE 'UFL'
Head Node
Jobs run as Normal Grid jobs These are not VOC nodes. Hence, these need not be VMs.

Figure 1.12: Virtual Organization Clusters showing participating (Clemson and Duke) and non-participating (UFL) sites. User submits jobs to his VO, which then gets scheduled to member sites. When resources are exhausted, other VOs are contacted to effect load balancing.

Virtual Organization can deploy its own customized environment- . Such environments will be scheduled to execute on the physical fabric, thereby permitting each VO to schedule jobs on its own private cluster. Figure 1.12 presents a use case of using Virtual Organization Clusters (VOCs) in a manner based on the operating principles of the Open Science Grid (HTTP://www.opensciencegrid.org) – a grid infrastructure known to support VOs instead of individual users. In the figure 1.12, there are two Grid sites. One site belongs to a VO and the other site does not. In this case, VO members Clemson and Duke provide nodes that form a Virtual Organization Cluster(VOC). Since these are participating members of a VOC, VMs are used to execute the user jobs. Once the resources in VO 'A' are exhausted, VOCs allow the use of resources from other grid sites that do not belong to the VO. Such sites are known as 'Non-participating sites'. Nodes acquired from such sites do not form VOC nodes and do not need to be virtual machines. The user jobs run as normal grid jobs but the nodes appear in the scheduler queue as other nodes that belong to VO 'A'.

Data transfers can be done through the OSG data-transfer mechanisms - (i.e. Phedex and dCache) and can use the GridFTP protocol. If a site becomes full, work can be migrated to another site using VM migration mechanisms. This use case represents an ideal form of grid operation, which would provide a homogeneous computing environment to the users. The Virtual Organization Cluster Model specifies the high-level properties of systems that support the assignment of computational jobs to virtual clusters owned by single VOs. Central to this model is a fundamental division of responsibility between the administration of the physical computing resources and the virtual machine(s) implementing each VOC. For clarity, the responsibilities of the hardware owners are said to belong to the Physical Administrative Domain (PAD). Responsibilities delegated to the VOC owners are part of the Virtual Administrative Domain (VAD) of the associated VOC. Each physical cluster has exactly one PAD and zero or more associated VADs.

In practice, Virtual Organization Clusters can be supplied by the same entity that owns the physical computational resource, by the Virtual Organizations (VOs) themselves, or by a contracted third party. Similarly, physical fabric on which to run the VOCs could be provided either by the VO or by a third party.

1.9.1 Physical Administrative Domain

The Physical Administrative Domain (PAD) contains the physical computer hardware (which comprises the host computers themselves, the physical network interconnecting those hosts, local and distributed storage for virtual machine images, power distribution systems, cooling, and all other infrastructure required to construct a cluster from hardware. Also within this domain are the host operating systems, virtual machine hypervisors, and central physical-level management systems and servers. Fundamentally, the hardware cluster provides the hypervisors needed to host the VOC system images as guests.

An efficient physical cluster implementation requires some mechanism for creating multiple compute nodes from a single VO-submitted image file. One solution is to employ a hypervisor with the ability to spawn multiple virtual machine instances from a single image file in a read-only mode that does not persist VM run-time changes to the image file. Another solution would be to use a distributed file copy mechanism to replicate local copies of each VM image to each execution host. Without this type of mechanism, the VO would be required to submit one VM image for each compute node, which would result in both higher levels of Wide Area Network traffic and greater administration difficulty.

1.9.2 Virtual Administrative Domain

Each Virtual Administrative Domain (VAD) consists of a set of virtual machine images for a single Virtual Organization (VO). A VM image set contains one or more virtual machine images, depending upon the target physical system(s) on which the VOC system will execute. In the general case, two virtual machine images are required: one for the head node of the VOC, and one that will be used to spawn all the compute nodes of the VOC. When physical resources provide a shared head node, only a compute node image with a compatible job scheduler interface is required.

VMs configured for use in VOCs may be accessed by the broader Grid in one of two ways. If the physical fabric at a site is configured to support both virtual head nodes and virtual compute nodes, then the virtual head node for the VOC may function as a gatekeeper between the VOC and the Grid, using a shared physical Grid gatekeeper interface as a proxy. In the simpler case, the single VM image used to construct the VOC needs to be configured with a scheduler interface compatible with the physical site. The physical fabric will provide the gatekeeper between the Grid and the VOC, and jobs will be matched to the individual VOC.

1.9.3 Provisioning and Execution of Virtual Machines

Virtual Organization Clusters are configured and started on the physical compute fabric by middleware installed in the Physical Administrative Domain. Such middleware can either receive a pre-configured virtual machine image (or pair of images) or provision a Virtual Organization Cluster on the fly using an approach such as In-VIGO [23], VMPlants [48], or installation of nodes via virtual disk caches [55]. Middleware for creating VOCs can exist directly on the physical system, or it can be provided by another (perhaps third-party) system. To satisfy VAD administrators who desire complete control over their systems, VM images can also be created manually and uploaded to the physical fabric with a grid data transfer mechanism such as the one depicted in the use case presented in Figure 1.12.

Once the VM image is provided by the VO to the physical fabric provider, instances of the image can be started to form virtual compute nodes in the VOC. Since only one VM image is used to spawn many virtual compute nodes, the image must be read-only. Run-time changes made to the image are stored in RAM or in temporary files on each physical compute node and are thus lost whenever the virtual compute node is stopped. Since changes to the image are non-persistent, VM instances started in this way can be safely terminated without regard to the machine state, since data corruption is not an issue. As an example, VM instances started with the KVM hypervisor are abstracted on the host system as standard Linux processes. These processes can be safely stopped.

Chapter 2

Related Work

There are a number of software that have been developed with the goal of designing efficient ways to store and lookup data in a distributed environment. The data could be neighboring node information(IP addresses, node IDs, running time, whether a node can be reached or not), user data or even routing information. The author now describes and compares some well know DHT libraries and P2P based systems.

2.1 Chord

The Chord project aims to build scalable, robust distributed systems using a peer-to-peer distributed hash lookup primitive. Chord is decentralized and symmetric (like most p2p algorithms), and can find data using log(N) messages, where N is the number of nodes in the system. Using the Chord lookup protocol, node keys are arranged in a circle. The circle cannot have more than 2m nodes. The circle can have ids/keys ranging from 0 to 2m-1. IDs and keys are assigned an m-bit identifier using what is known as consistent hashing. The SHA-1 algorithm is the base hashing function for consistent hashing. The consistent hashing is integral to the probability of the robustness and performance because both keys and IDs (IP addresses) are uniformly distributed and in the same identifier space. Consistent hashing is also necessary to let nodes join and leave the network without disrupting the network[62].

Each node has a successor and a predecessor. The successor to a node or key is the next node in the identifier circle when you move clockwise. The predecessor of a node or key is the next node in the id circle when you move counter-clockwise. A logical ring with positions numbered 0 to 2^(m-1) is formed among nodes. Key k is assigned to node successor(k), which is the node whose identifier is equal to or follows the identifier of k. If there are N nodes and K keys, then each node is responsible for roughly K / N keys. When the (N+1)th node joins or leaves the network, responsibility for O(K/N) keys changes hands[62]. Each node knows the IP address of its successor. If each node knows the location of its successor, you can perform linear search over the network for a particular key. This is a naive method for searching the network. Chord does not use DHT for routing as others do. The issues that arise therefore will be discussed below.

2.2 Pastry

Pastry provides the following capabilities. First, each node in the Pastry network has a unique, uniform random identifier (nodeId) in a circular 128-bit identifier space. When presented with a message and a numeric 128-bit key, a Pastry node efficiently routes the message to the node with a nodeId that is numerically closest to the key, among all currently live Pastry nodes. The expected number of forwarding steps in the Pastry overlay network is O(log N)(similar to Chord), while the size of the routing table maintained in each Pastry node is only O(log N) in size (where N is the number of live Pastry nodes in the overlay network). At each Pastry node along the route that a message takes, the application is notified and may perform application-specific computations related to the message[17].

Pastry used DHT for routing. This ensures that the system is scalable and fault tolerant. The routing metric is supplied by an external program based on the IP address of the target node and this means that the metric can be easily switched to shortest hop count, lowest latency, highest bandwidth, or even a general combination of metrics. The hash table's keyspace is taken to be circular, like the keyspace in the Chord system, and node IDs are 128-bit unsigned integers representing position in the circular keyspace. Node IDs are chosen randomly and uniformly so peers who are adjacent in node ID are geographically diverse[61]. The routing overlay network is formed on top of the hash table by each peer discovering and exchanging state information consisting of a list of leaf nodes, a neighborhood list, and a routing table. The leaf node list consists of the L/2 closest peers by node ID in each direction around the circle. Pastry also employs neighborhood lists and routing tables to locate the nearest peers.

2.3 Kademlia

Kademlia uses a "distance" calculation between two nodes. This distance is computed as the exclusive or of the two node IDs, taking the result as an integer number. Keys and Node ID's have the same format and length, so distance can be calculated among them in exactly the same way. The node ID is typically a large random number that is chosen with the goal of being unique for a particular node. It can and does happen that nodes from Germany and Australia are "neighbors"; they have chosen similar random node IDs[51]. Kademlia uses Exclusive-OR because of its properties with geometric distance formula including the distance between a node and itself is zero and the symmetric property i.e. the "distance" calculated from A to B and from B to A are the same. These properties provide most of the important behaviors of measuring a real distance with a significantly lower amount of computation to calculate it. Each Kademlia search iteration comes one bit closer to the target. A basic Kademlia network with 2n nodes will only take n steps (in the worst case) to find that node.

Kademlia routing tables consist of a list for each bit of the node id. (e.g. if a node ID consists of 128 bits, a node will keep 128 such lists.) A list has many entries. In Kademlia jargon, these lists are called buckets. Every entry in a list holds the necessary data to locate another node. The data in each list entry is typically the IP address, port, and node id of another node. Every list corresponds to a specific distance from the node. Nodes that can go in the nth list must have a differing nth bit from the node's id; the first n-1 bits of the candidate id must match those of the node's id. This means that it is very easy to fill the first list as 1/2 of the nodes in the network are far away candidates. The next list can use only 1/4 of the nodes in the network (one bit closer than the first), etc[14]. With an ID of 128 bits, every node in the network will classify other nodes in one of 128 different distances, one specific distance per bit. As nodes are encountered on the network, they are added to the lists. This includes store and retrieval operations and even when helping other nodes to find a key. Every node encountered will be considered for inclusion in the lists. Therefore the knowledge that a node has of the network is very dynamic. This keeps the network constantly updated and adds resilience to failures or attacks.

2.4 Brunet (Based on Symphony)

Brunet is a robust DHT library that is based on the Symphony[50] DHT peer-to-peer algorithm. Like Chord, the identifier space in Symphony is constructed as a ring with a perimeter of one unit. When a node joins the ring, it chooses randomly a real number in the rank [0,1). This node will handle the identifier rank between this id and its immediate predecessor's id. On the other hand, each node maintains a pointer to its successor and predecessor on the ring. However, the finger table of Chord is replaced by a constant (configurable) number of long distance links. These links are chosen randomly according to harmonic distributions (hence the name Symphony), which favors large distances in the identifier space in systems with few nodes, and smaller distances as the system grows.

Brunet[26] can therefore handle several problems that exist today in networking systems. Network transports are abstracted as Edge objects and TCP, UDP and TLS/SSL can be used as modes of transportation. The UDP NAT Traversal is completely distributed. Brunet also implements a ring type topology for routing that is similar to Chord. Brunet also has DHT implemented and hence deals with fault tolerance and quick lookups for stored information which could include node location. Brunet also supports packet based and RPC based communication primitives. Finally, it also employs a distributed tunneling system to maintain topology in the presence of some routing difficulties (untraversable NATs, BGP outages, firewalls, etc.).

IPOP or Internet Protocol over Peer-to-Peer [42], is a tunneling software that uses a peer-to-peer architecture rather than a client-server architecture. IPOP Brunet [26], a software library for P2P networking written in C# and developed using Mono. It's key features include:

1. Network transports are abstracted as Edge objects. Thus enabling TCP, UDP and TLS/SSL transports,
2. Completely distributed UDP NAT traversal,
3. Implementation of Chord/Kleinberg/Symphony type ring topology for routing,
4. A complete DHT implementation and
5. Distributed tunneling system to maintain topology in the presence of some routing difficulties (- NATs, BGP outages, firewalls, etc.).

IPOP [42] is a self-configuring IP-over-P2P virtual network overlay which provides the capability for nodes behind NATs and some firewalls to all appear to be in the same subnet. Besides being transparent to applications, IPOP was deployed in the VOC [53] model for the reasons explained in Section 1.8.

2.5 Work Related to VOCs

Virtualization was discusses in detail in 1.4. [37] Virtualization of grid systems was first proposed in [38] and provides much required abstraction and usability to grid users and system administrators. In a virtualized environment, access can be granted as per the administrators needs. This reduces the potential harm that can be done to the hardware. Hardware multiplexing is also possible; the same hardware can be used to host different kinds of Virtual Machines. [38] Higher-level system components may also be virtualized; examples include virtual networking systems such as ViNe [65], VNET [63], VDE [33], and IPOP[42].

Globus Virtual Workspaces, implemented as part of the Globus Nimbus toolkit, provide a lease-oriented mechanism for sharing resources on grid systems with fine-grained control over resource allocation. A Virtual Workspace is allocated from a description of the hardware and software requirements of an application, allowing the workspace to be instantiated and deployed on a per-application basis. Another lease-oriented mechanism for grid system virtualization is Shirako [46], which allows resource providers and consumers to negotiate resource allocations through automated brokers. Back-end cluster provisioning is provided by Cluster-On-Demand (COD) [29], an automated system that performs rapid re-installation of physical machines or Xen virtual machines [45]. Dynamic Virtual Clustering allows clusters of virtual machines to be instantiated on a per-job basis for the purpose of providing temporary, uniform execution environments across clusters co-located on a single research campus. These clusters comprise a Campus Area Grid (CAG), which is defined as “a group of clusters in a small geographic area ... connected by a private, high-speed network” [35].

Virtual Organization Clusters (discussed in detail in Section 1.9)[53] differ from explicit leasing systems such as Globus Nimbus and Shirako, in that virtual clusters are leased autonomically through the use of pilot jobs, which provide for dynamic provisioning in response to increasing workloads for the associated VO [54]. VOCs remain transparent to end users and to non-participating entities, while enabling participating VOs to use overlay networks, such as IPOP [42], to create virtual environments that span multiple physical domains. Unlike Campus Area Grids and VioCluster environments, however, these physical domains are grid sites connected via low-bandwidth, high-latency networks. These unfavorable connections, coupled with overheads introduced by the addition of virtualization systems, make VOCs better suited to high-throughput, compute-bound applications than to high-performance applications with latency-sensitive communications requirements [36].

Chapter 3

Proposed System

3.1 Overlay Networking Details

Virtual networking systems that enable VOCs to span multiple resources must be able to:

1. Support an ad-hoc network, i. e. support the dynamic addition and removal of nodes based on workload,
2. Load balance between multiple servers efficiently and without user involvement,
3. Assign IP addresses dynamically,
4. Route traffic efficiently between machines in the pool,
5. Have the capability to traverse NAT and firewalls.

IPOP or Internet Protocol over Peer-to-Peer was discussed in detail in 1.8. As mentioned in Chapter 1, modern day research and education requires compute resources that can no longer be confined to a definite pre-determined size. While the number of Super Computing sites have increased ten folds in the last two decades [19], access to these resources have been limited due to various factors including availability, cost and politics. Domain scientists desire as much resources as they can acquire so that their algorithms run as efficiently as possible. They are not concerned about how their jobs are scheduled, what machines their jobs run on or how the resources are allocated. If these scientists and other users had to be concerned with the design and administration of all the resources they required, their research and subsequently use of these systems would be greatly hampered.

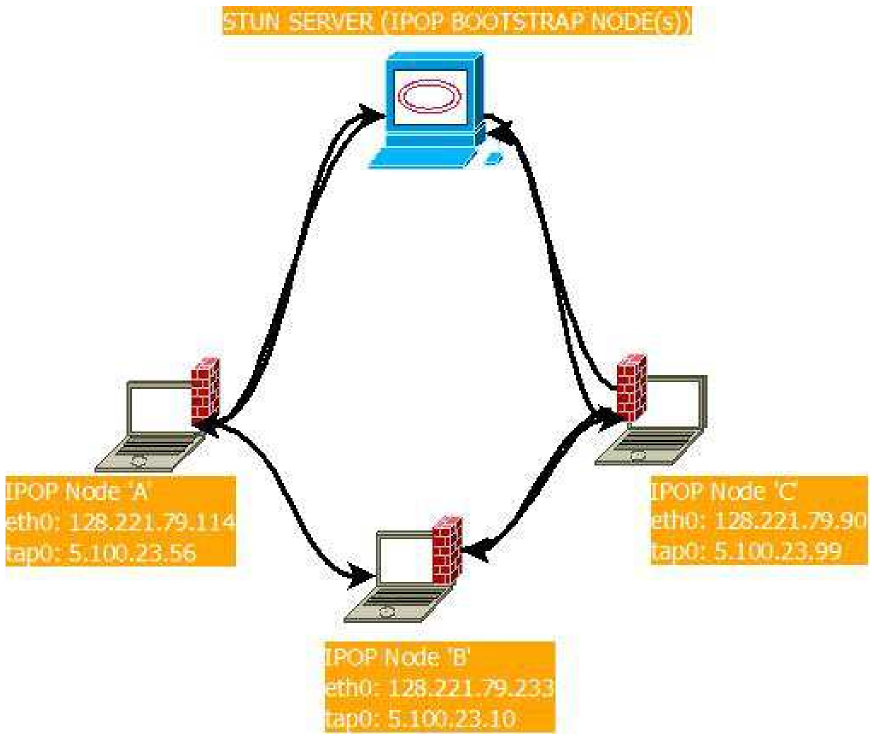


Figure 3.1: IPOP as a stun server: The 3 nodes A, B and C have publicly routable IP addresses on network interface eth0. But they cannot communicate with each other as they are all behind NATs. The can, however, communicate with other machines on the Internet including the IPOP bootstrap node that is shown. This bootstrap node(s) functions as a STUN server. It allocates addresses to the three nodes A, B and C on the virtual network interface tap0 (tap device). Therefore, all three nodes can communicate effectively with each other.

The main objective for researching and using a DHT based peer-to-peer networking system lies in the usability of the system and efficiency of acquiring(removing) additional resources. These resources could be storage, computing power (additional nodes, powerful workstations) or even nodes used for improving routing. These aspects will be examined in detail. Traditional Client-Server architectures that were not built on top of overlays have the difficulties in load balancing, inability to traverse NATs (therefore, inability to be used effectively in modern WAN environments) and bad fault tolerance.

The overlay system adopted here was based on Brunet. This is therefore a DHT based P2P algorithms that uses an overlay network. The tunneling and routing system developed on top of Brunet is IPOP (Discussed in detail in 1.8). The IPOP system is setup around two kinds of nodes - Bootstrap nodes and IPOP nodes (also known as p2p nodes). If there are two nodes behind separate firewalls or behind a NAT[18], they cannot communicate directly. NAT (Network Address Translation) was introduced as the IANA (Internet Assigned Numbers Authority) was running out of 32 bit IP addresses to allocate users of the Internet[18]. In order for two nodes behind a NAT to communicate they must use a STUN[20] server or an intermediate machine that has access to the subnets that both nodes belong to.

The following methods delineate the steps involved in overlay communication setup in the VOC model for two nodes behind a firewall or NAT:

1. IPOP uses a bootstrap node(s) as a backbone for communication. These nodes must have publicly routable IP addresses to be able to effectively serve as STUN servers and intermediary machines. The initial routing and access to the nodes inside the overlay are facilitated by the bootstrap nodes. These nodes also act as DHCP servers and assign addresses to the nodes that connect to them.

2. The IPOP or P2P nodes that need to communicate must be aware of the IP addresses of the bootstrap nodes. The bootstrap nodes are setup with the help of the IPOP Bootstrap software/daemon.

3. Before continuing, the IPOP namespace, DHCP address range, subnet mask and certificate information must be inserted into the DHT. This DHT is distributed among the bootstraps and later among the nodes once they joint the overlay.

4. TAP devices are used as a means of communication. TAP or network tap simulates an Ethernet

device and it operates with layer 2 packets such as Ethernet frames. TAP is used to create a network bridge.

5. Once the TAP device is setup on the node, the IPOP Client software/daemon proceeds to contact the DHCP server on the bootstrap to get an IP address. All IPOP nodes use a namespace as a unique group identifier to communicate with the nodes that belong to the same private group. This will be explained more in the next section.

6. Once the IP Address is acquired by the nodes, the routing is complete. The neighboring nodes are notified of the new node address and unique identifier. The DHT is updated and gets more distributed as the number of nodes increase.

3.2 IPOP Bootstraps and Namespaces in Clemson

```
<?xml version="1.0"?>
<DHCPConfig>
<Namespace>cirg_cpsc362</Namespace>
<IPBase>5.130.127.0</IPBase>
<Netmask>255.255.255.0</Netmask>
<LeaseTime>7200</LeaseTime>
</DHCPConfig>
```

Figure 3.2: IPOP DHCP Configuration File

```
<?xml version="1.0"?>
<NodeConfig>
  <BrunetNamespace>ufl_test0_15</BrunetNamespace>
<EdgeListeners>
<EdgeListener type="tcp">
 <port>0</port>
</EdgeListener>
<EdgeListener type="udp">
<port>0</port>
</EdgeListener>
</EdgeListeners>
<RemoteTAs>
<Transport>brunet.tcp://131.114.53.188:49301</Transport>
<Transport>brunet.udp://131.114.53.188:49301</Transport>
<Transport>brunet.tcp://195.116.60.97:49301</Transport>
<Transport>brunet.udp://195.116.60.97:49301</Transport>
<Transport>brunet.tcp://192.16.125.11:49301</Transport>
 </RemoteTAs>
 <XmlRpcManager>
 <Enabled>true</Enabled>
  <Port>10000</Port>
 </XmlRpcManager>
 <NCService>
 <Enabled>true</Enabled>
  <OptimizeShortcuts>true</OptimizeShortcuts>
 <Checkpointing>false</Checkpointing>
 </NCService> </NodeConfig>
```

Figure 3.4: IPOP Bootstrap Configuration File

```
<?xml version="1.0"?>

<IpopConfig>

<IpopNamespace>cirg_cpsc362</IpopNamespace>

<VirtualNetworkDevice>tapipop</VirtualNetworkDevice>

<EnableMulticast>true</EnableMulticast>

<EndToEndSecurity>false</EndToEndSecurity>

<DHCPPort>67</DHCPPort>

<AllowStaticAddresses>true</AllowStaticAddresses>

</IpopConfig>
```

Figure 3.3: IPOP Namespace Configuration File

```
<?xml version="1.0"?>
<NodeConfig>
  <BrunetNamespace>ufl_test0_15</BrunetNamespace>
<EdgeListeners>
<EdgeListener type="tcp">
   <port>0</port>
</EdgeListener>
<EdgeListener type="udp">
     <port>0</port>
</EdgeListener>
</EdgeListeners>
<RemoteTAs>
<Transport>brunet.tcp://131.114.53.188:49301</Transport>
<Transport>brunet.udp://131.114.53.188:49301</Transport>
<Transport>brunet.tcp://195.116.60.97:49301</Transport>
<Transport>brunet.udp://195.116.60.97:49301</Transport>
</RemoteTAs>
 <XmlRpcManager>
  <Enabled>true</Enabled>
  <Port>10000</Port>
 </XmlRpcManager>
 <NCService>
 <Enabled>true</Enabled>
   <OptimizeShortcuts>true</OptimizeShortcuts>
  <Checkpointing>false</Checkpointing>
</NCService>
</NodeConfig>
```

Figure 3.5: IPOP Node Configuration File

IPOP facilitates the use of three kinds of bootstrap infrastructure:

- Local Bootstraps: These are a set of publicly routable bootstrap nodes that could be behind an institutional firewall. Such an infrastructure is recommended when all IPOP clients are confirmed to be behind the same firewall.

- Public Bootstraps: These are a set of publicly routable bootstrap nodes that are not behind any firewall and can be used to interconnect IPOP nodes from different geographic locations and network domains.

- Combo Bootstraps: This infrastructure combines a set of local and public bootstraps that can be accessed from within or without a firewall. Such a setup is used when a larger multi-institutional IPOP client/node environment is required.

All of the above have been used in varying extent and a detailed description of their setup is found in Chapter 4. The IPOP Namespaces are group identifiers that are similar to the broadcast address in a network. But there are some key differences here. These namespaces are used to support the Virtual Networks that form the overlay. IPOP supports multiple Virtual Networks sharing the same overlay as follows:

- Each virtual IP is associated with a namespace, e.g., IPOP views an IP in the form Namespace:IP. This is transparent to the user and to the virtual network interface - each IPOP router is configured to be bound to a single IP namespace.

- Dynamic IP-to-P2P address allocation and resolution through the use of a decentralized data store, i.e., the DHT mentioned above

The setup in Clemson involved separate namespaces that were used in different virtual environments as discussed in Chapter 4. Each namespace had a different set of DHCP address ranges and clients, but they could all be setup to use the same set of bootstrap nodes. This meant that the same group of IPOP bootstrap nodes could be used to support a large number of virtual networks and the nodes on different networks cannot interfere with nodes in other networks.

3.3 Multi-site communication and Watchdog

Condor[64] was used to perform the VM Scheduling on-site as well as the job scheduling among the sites. Condor is able to handle the queuing mechanism, priority schemes as well as scheduling policies. Site policies can vary and the VOC model has the ability to work around different policies. In order to perform jobs using resources across different grid sites, Globus[40] was used. Globus has four main components:

- Authentication - GSI(Grid Security Infrastructure)
- Resource management - GRAM(Grid Resource Allocation and Management)
- Data transfer - GridFTP(Grid File Transfer Protocol)
- Resource discovery and monitoring - MDS(Monitoring and Discovery Service)

Delving further into the 'Resource Management' phase, we can understand how two entities that belong to different grid domains can share resources with each other. GRAM uses a daemon that runs on the Compute Element or Head node of every Cluster(Grid site). This daemon that runs on grid resources is called a Gatekeeper and has the following features:

- Processes incoming Globus requests
- Authenticates Users Configured to trust a given CA
- Maps user to local account DN to username grid-mapfile
- Passes the job onto the jobmanager

In the last stage of the resource management, Globus hands the job over to the jobmanager (in our case Condor) through the Gatekeeper. But when should this be done? What sites should be contacted? what resources should be requested? How long should they be used? what happens if an allocated resource fails? To answer these questions and more, a software/daemon called a Watchdog was developed. The Watchdog performs the following:

1. Monitors job queue for running jobs and queued jobs
2. Based on VMs to Jobs ratio, computes the requirement for booting or shutting down allocated nodes.

3. Communicates to the Gatekeepers that are deployed on the sites that share resources with another site. (Ideally only one watchdog is required at the head node of the user's VO - discussed in 4)

A Greedy algorithms was used for most of the experiments and testing. The main reason for this is that other self-learning algorithms did not improve much due to the scheduling intervals that Condor uses. The main steps of this Greedy algorithm have been described below:

- Invoke PROBE_STATE, a module to determine the number of jobs in the schedule's queue,
- Examine the jobs to VMs ratio and enter a the appropriate state, described below,
- Take an action based on the current state,
- Wait for a given interval (10 seconds in the prototype).

3.4 Condor IPOP Pool - Local/Single Site

The Engage VO gets job submissions of varying sizes that require compute resources for varying amounts of time. This implies that there may be certain jobs that require a single node and yet again other jobs that require all the available nodes(in this case 16 VMs. The setup of the physical cluster is explained in detail in Chapter 4). This is where the the Watchdog daemon starts to play out its role. The Watchdog as mentioned before resides on the Head Node which also has the Gatekeeper. The Condor Central Manager will reside on the Head node too and is used to schedule incoming jobs on this local site. A tap device has been setup on this Central Manager so that it can acquire an IPOP address from the bootstrap(infrastructure) that it has been connected to. Once a job has been submitted to the Engage VO, it will appear in the job queue of the Central Manager. The Watchdog, which uses a greedy algorithm, monitors the queue and the Jobs-to-VM ratio. If this ratio exceeds '1' for a period of 10-20 seconds(this can be modified), the job was unable to be scheduled by the job manager on any of the presently running/executing nodes. The watchdog then proceeds to start a new node, monitoring the Jobs-to-VM ratio ensuring the maximum site capacity is not exceeded. The watchdog also has the capability to shut down nodes if the ratio decreases below '1' for a certain period of time.

A single IPOP Bootstrap is able to handle a thousand p2p nodes; although we have been able to test only with a hundred nodes. In this case, we had a single bootstrap node which had a public

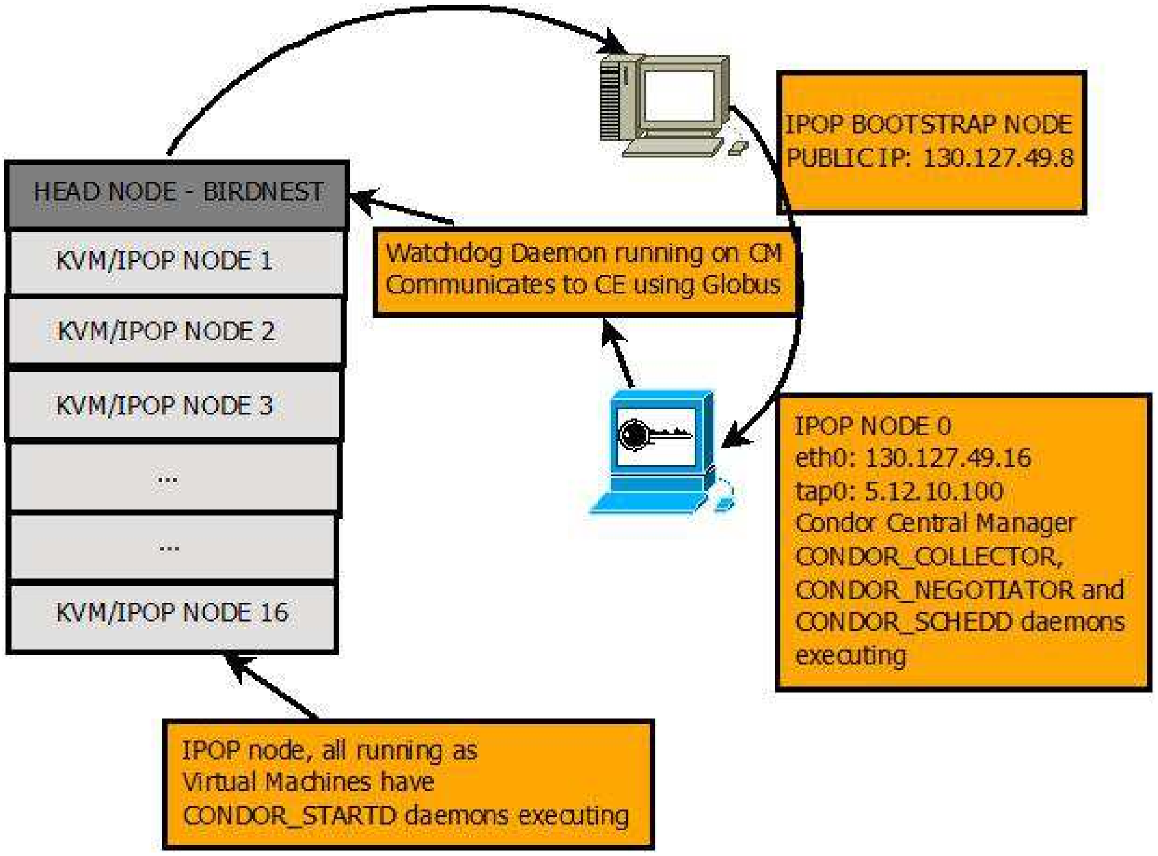


Figure 3.6: The Single Site Setup. This is how a VO member site would look with the overlay setup. One or more local IPOP bootstraps would be present. In this experiment, a single bootstrap with a public IP address. Another machine (Virtual or Physical) setup outside the cluster would be the Condor Central Manager. This node would also be an IPOP node and hence communicate with other IPOP nodes through the TAP device. This Watchdog daemon sends pilot jobs through Globus to start and stop VMs on the Cluster based on work-loads. The machines appear and disappear in the schedulers pool which is accessible to the user who needs to monitor the state of his job(s) on the VO.

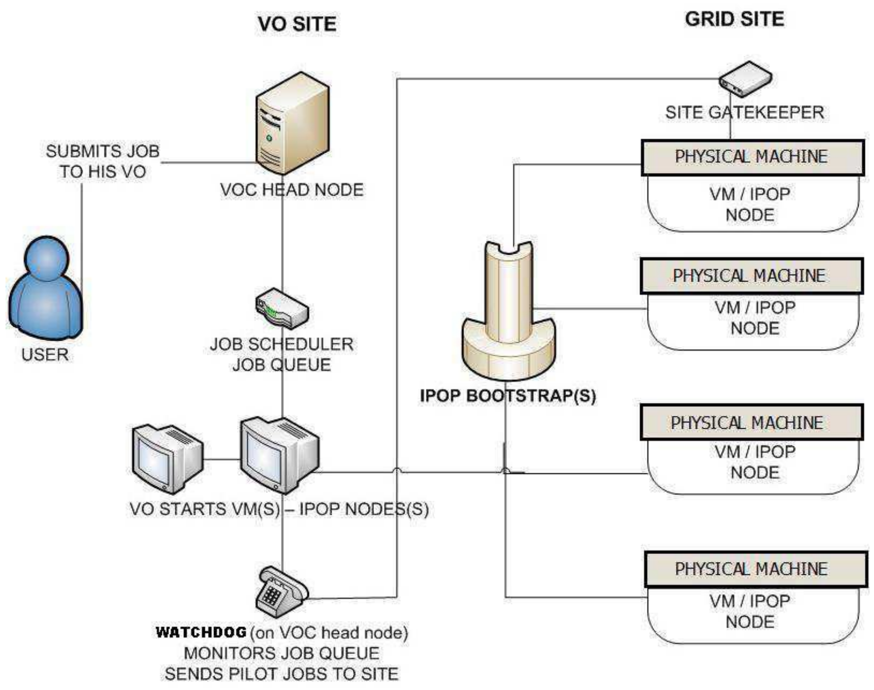


Figure 3.7: VOC over a single site

IP address. The DHCP address range, subnet, lease period and namespace need to be specified in a config file before the bootstrap can be started. This bootstrap was a workstation which was a dedicated machine, even though this is not a requirement. As mentioned in 3.2, local, public or even a combo bootstrap infrastructure can be used. But as single sites only employ local nodes, a local IPOP bootstrap infrastructure(more than one bootstrap) may only be required. As mentioned above, the Central Manager is connected to the bootstrap and hence has an IPOP IP-address on the virtual tap device. The watchdog starts a new VM using standard KVM/Qemu command line tools and these new nodes acquire an IPOP IP-address from the DHCP server running on the bootstrap. It is hence able to join the local Condor pool running.

The new nodes are bought up in snapshot-mode. This annuls the need to stage the multi-gigabyte VM image across all the nodes. This is done efficiently using a distributed File system such as PVFS(explained in 3.5). The watchdog monitors the Jobs-to-VM ratio, starting and shutting down nodes. As jobs get executed, the machines start leaving the pool. This, therefore, is how dynamic resizing of resources is done to:

1. Ensure constant availability of compute resources

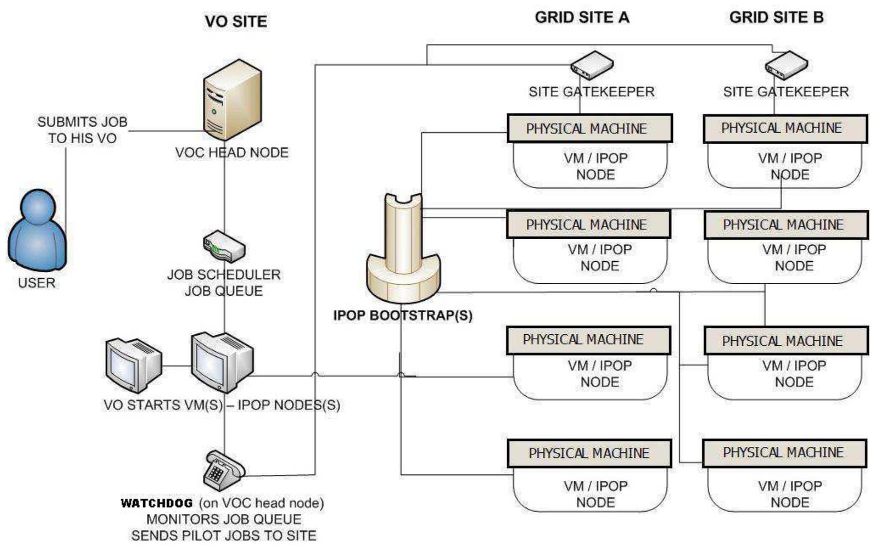


Figure 3.8: VOC over Multiple Grid Sites using an Overlay Network

2. Improve System utilization
3. Improve execution time and efficiency of algorithms
4. Conserve energy by using resources only when they are required
5. Provide load balancing and hence fault tolerance

3.5 Condor IPOP Pool - Multiple Grid Sites

A couple of key questions need to be answered when resource sharing across multiple grid sites needs to be accomplished.

1. How can nodes belonging to different grid administrative sites effectively and securely communicate?
2. How can the multi-gigabyte node/VM image be securely and efficiently transferred to multiple sites? Once transferred, how can this image be used across the physical infrastructure of individual sites?
3. How can firewalls or NATs be traversed without user or administrative annoyance? (this annoyance could be in terms of changes that need to be made or more important, security)

The first two queries can be solved partially using Globus. As mentioned in Section 3.3, in the resource management phase(GRAM), Globus uses a daemon called the Gatekeeper that is deployed on the Head node of a every site. The Watchdog (that has been designed in the VOC model and explained in Section 3.4) communicates between these sites using the Gatekeeper. A Virtual Organization can have multiple grid sites as participating members. These members allocate time and resources to different job submissions from the VO. All the members of a VO have been securely authorized for multi-grid allocations. It is possible, by configuration modifications, for sites belonging to different VOs to share resources too. Therefore, with the use of the VOC model incorporating a p2p based overlay network(as described in Chapter 4), a huge number of compute resources (in the ten thousands perhaps) can be made available to a single VO user to efficiently load balance between sites.

Only one VM/node image is required per site. Therefore, using GridFTP (a part of Globus) the image can be transferred to the either the Head node or another storage server on the site. Once it is there, a distributed file system can be used to use the new images. The file system used here is PVFS[28] for its massive scalability and ease of administration. Therefore, when a Watchdog deployed at one site contacts the gatekeeper at another, the same watchdog has the authority to start up new nodes in the snapshot mode. This is much more efficient in time, space and management, rather having to move the multi-gigabyte image to the disk of every machine.

Finally, question 3 can be answered with the help of an overlay networking system, such as the one used here. IPOP was explained in 1.8 and has the capability of NAT and Firewall traversal. This is established with the help of bootstrap nodes and IP tunneling. All the information that is transferred within an IPOP network is encrypted and hence secure. By using this system, administrators do not have the hard task of assigning IP addresses. This can be a difficult task when a large number of machines in different sites and geographical locations are involved. Administrators do not have to be concerned with resource allocation either, as jobs get scheduled using Condor. The Watchdog also takes care of resizing the machine pool, therefore starting and shutting down is also handled dynamically. Finally, domain scientists and other end users do not need to learn new protocols or software in order to run their algorithms with the maximum amount of resources that can legally be acquired.

Chapter 4

Experiments & Results

4.1 Building the Physical and Virtual Clusters

Clemson University's Cyber Infrastructure Research Group is an Open Science Grid(OSG) production site. More information on OSG and its operations can be found in [57] and [59]. OSG supports a number of Virtual Organizations such as Engage, Nanohub, amongst many others. In OSG, a Virtual Organization (VO) refers primarily to a collection of people (members of the VO) and also includes the group's computing/storage resources and services (e.g., databases). The terms site, computing element (CE), and/or storage element (SE) are used often to refer to the resources owned and operated by a VO[22]. In order to efficiently test performance and overheads, the primary physical cluster was build as an OSG site that supports multiple VOs. This ensured that real jobs submitted by OSG users and synthetic work loads were used in the testing of the VOC model over IPOP.

The Physical Cluster included one Head Node and 16 Compute Nodes. The physical configuration of all the machines were similar being Dual-core Intel Xeon 3070 CPU(s) with 4 binary gigabytes (GiB) of Random Access Memory (RAM). All the machines had the Kernel Virtual Machine installed for virtualization. The primary operating system used was the Red Hat based CentOS 5.2 with PVFS as the distributed file system(as described earlier). The Head node had Globus installed and as mentioned in 3.3, the gatekeeper kept track of all incoming communications, authorizing jobs and users based on VO specification. The Engage VO was used primarily for all tests. A Single Core on each compute node was dedicated to the Engage VO; thus a total of 16 VMs were allotted for the VO. When a scientist(user) submits a job, the watchdog monitors the job queue on the head node(which also is a DHCP server). As the need requires, the watchdog starts a VM and the acquires an IP address. The jobs gets scheduled, executes on a node and then shuts down once completed. Please note that at this stage there is no overlay network. The DHCP addresses, leases, network card MAC addresses have to be hard coded and is not scalable beyond the site.

The Virtual Cluster included resources that were either lease-based(such as Amazon EC2[24]) or not lease-based(such as resources from other VO members). As mentioned in 3.1, VOCs do not adhere to any particular networking standard, protocol or architecture. Even though certain core requirements need to be satisfied in order to ensure that the definition of the model is not lost (these were discussed in 3.1). IPOP was chosen because it's DHT based P2P architecture has high scalability, ease of administration and fault tolerance. The basic virtual cluster on-site was based on top of the physical cluster explained above. The Engage VO was allotted 16 virtual machines as computer resources. But in the case of the virtual cluster, the networking administration is handled by the IPOP bootstrap and not the Cluster head node as in the previous case. The watchdog was modified to contact the gatekeepers of other participating member sites(or amazon instances in the case of EC2). A common image that was pre-configured with necessary applications and config files to communicate with the IPOP bootstrap was transferred to the other site(s). As the machines were booted(based on workload), they joined the Condor-IPOP pool forming a Virtual Cluster. This Cluster has the capability of scaling up to large numbers and dynamically re-size based on the work load.

4.2 Local Testing

4.2.1 Experiments

The overhead of adding virtualization to grid systems has previously been evaluated in [36]. The overhead of using the IPOP network overlay has been independently studied in [42] and the effect of using both a grid system and an overlay on the top of the Virtual Organization Cluster[53] model has been published in [52]. Since the overhead of virtualization would not primarily affect job service time in a grid system with sufficient physical resources to handle all jobs concurrently, a chief concern was the amount of latency that might be added by the overlay scheduling system. In order to ensure that this overlay overhead would not have unexpected detrimental impacts on compute-bound jobs running within the virtual machines, tests were conducted using an Open Science Grid (OSG) [59] site configured to support virtualization. The OSG grid site was configured with 16 dual-core compute nodes, each with an Intel Xeon 3070 CPU and 4 binary gigabytes (GiB) of Random Access Memory (RAM), with the Kernel-based Virtual Machine (KVM) [60] hypervisor running within a 64-bit installation of CentOS 5.2. Virtual machine images were configured with 32-bit CentOS 5.2 and located on a shared Parallel Virtual File System (PVFS) [28] store. Virtual machine instances were booted directly from the image located on the shared file system, without the need of staging the image to the local compute nodes. This was accomplished using the "snapshot" mode of KVM(as explained earlier). These shared images were made available in a read-only configuration, with non-persistent writes redirected to a temporary file on local disk storage at each compute node. Internet connectivity for the test site was provided by an edge router using Network Address Translation (NAT), with the physical compute nodes isolated in a private IPv4 subnetwork. OSG connectivity was provided through the standard OSG software stack including Globus [40].

A VOC head node was constructed using a virtual machine hosted by an off-site laboratory workstation. Condor [64] was installed on the head node to serve as a scheduler, and synthetic workload jobs were submitted directly to the VOC local pool. A watchdog daemon process, using the naive greedy watchdog algorithm with added support to maintain a minimum number of running VMs at all times, was run on the same head node. This watchdog created pilot jobs, which were submitted through a Globus client to the OSG test site, in response to the arrival of synthetic workload jobs in the VOC Condor queue. The pilot jobs started virtual compute nodes, which joined an IPOP network anchored at the workstation by contacting its bootstrap service. Once connected to the private IPOP network, the virtual compute nodes joined the Condor pool created by the collector on the VOC head node (the workstation-hosted virtual machine, also joined via IPOP), and Condor scheduled and executed the actual test jobs. Whenever the watchdog daemon determined that an excess number of pilot jobs were running in comparison to the size of the Condor queue, the pilot jobs were instructed to terminate, causing the virtual machines to be terminated.

To measure the relative performance difference of using a VOC with overlay scheduling and IPOP overlay networking, two sets of tests were conducted. A synthetic workload consisting of a 10-minute sleep procedure was devised, in order to approximate compute-bound jobs without incurring potential variations in service times that could result from running an actual compute-bound job within a virtual machine. In the control trials, a batch of 50 sleep jobs was submitted directly to

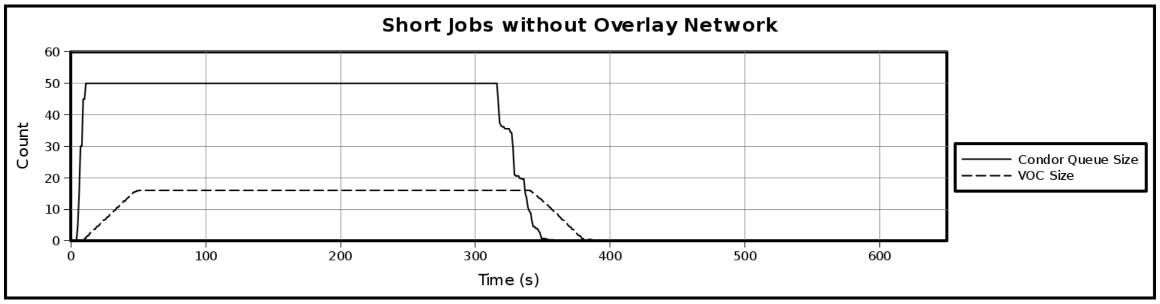


Figure 4.1: Short Jobs without the Overlay Network

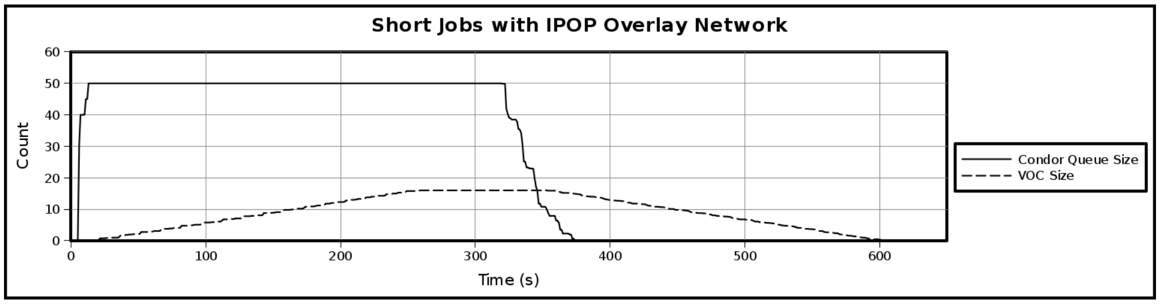


Figure 4.2: Short Jobs with the Overlay Network

the local scheduler on the physical grid site head node. For the experiment trials, the same batch of 50 jobs was submitted directly to the Condor central manager running within the VOC. Total makespan times were collected for both sets of trials, and each trial was repeated 10 times to reduce the effects of random variation in observed makespan lengths. Descriptive and relative statistics were computed for the makespan times. Throughput measures in jobs per second were also computed.

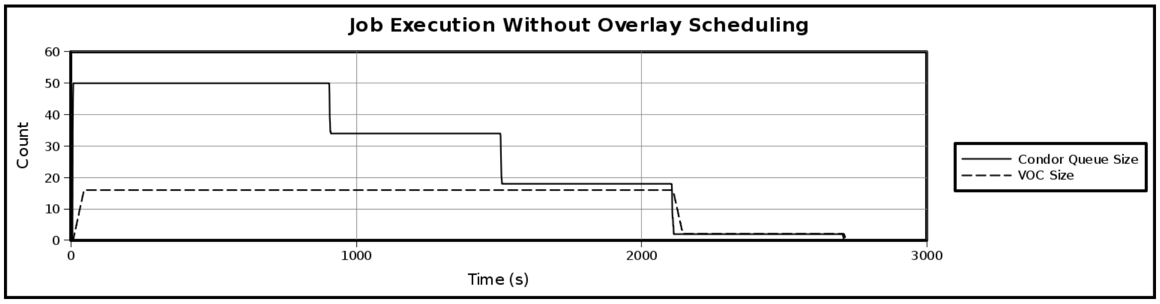


Figure 4.3: Long Jobs without the Overlay Network

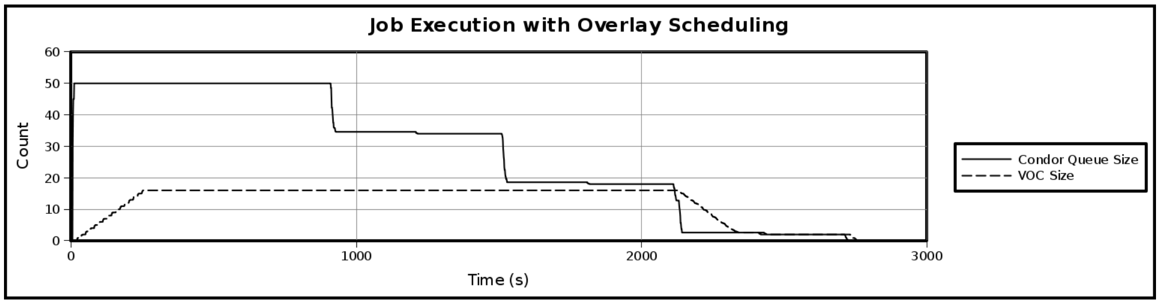


Figure 4.4: Long Jobs with the Overlay Network

Table 4.1: Observed makespan lengths and system throughputs for 10 overlay experiment trials

Test	Without Overlay	With Overlay	Change (Absolute)	Change (Relative)
Minimum Makespan (s)	2706	2714	8.000	0.2956 %
Median Makespan (s)	2709	2720	11.00	0.4061 %
Maximum Makespan (s)	2710	2737	27.00	0.9963 %
Mean Makespan (s)	2708	2722	13.50	0.4985 %
Makespan Standard Deviation (s)	1.414	7.735	6.321	447.0 %
Minimum Throughput (Jobs $\cdot$ s^{-1})	1.845×10^{-2}	1.827×10^{-2}	-1.820×10^{-4}	- 0.9865 %
Median Throughput (Jobs $\cdot$ s^{-1})	1.846×10^{-2}	1.839×10^{-2}	-7.467×10^{-5}	- 0.4045 %
Maximum Throughput (Jobs $\cdot$ s^{-1})	1.848×10^{-2}	1.842×10^{-2}	-5.447×10^{-5}	- 0.2948 %
Mean Throughput (Jobs $\cdot$ s^{-1})	1.846×10^{-2}	1.837×10^{-2}	9.000×10^{-5}	- 0.4954 %
Throughput Standard Deviation	9.644×10^{-6}	5.207×10^{-5}	4.243×10^{-5}	439.9 %

4.2.2 Results

Results of the trials, summarized in table 4.1, indicated a slight increase in average makespan time (less than one half of one percent) for jobs submitted through the overlay scheduling system, compared to jobs submitted directly to the physical cluster scheduler. This increased makespan length corresponded to a similarly small decrease in job throughput resulting from the addition of the overlay. In the worst case observed in all trials, the maximum makespan and minimum throughput were affected by less than one percent. Variations in makespan and throughput observations between trials was substantially increased by over 400% when the overlay scheduler and network were added, likely due to the addition of a second layer of interval scheduling with the additional Condor pool overlaid on top of the physical Condor pool. Plotted traces of mean observations (figures 4.3 and 4.4) further confirmed the minimal overhead of the VOC overlay system.

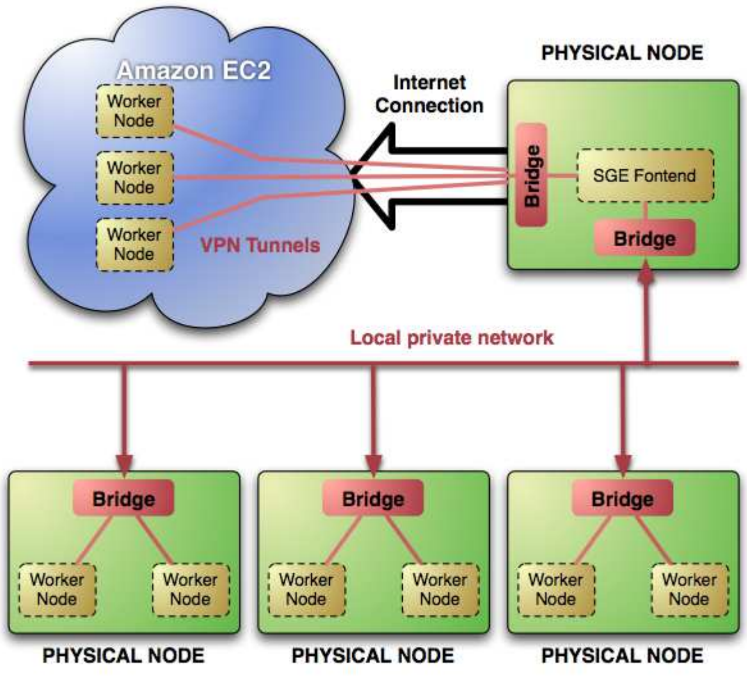


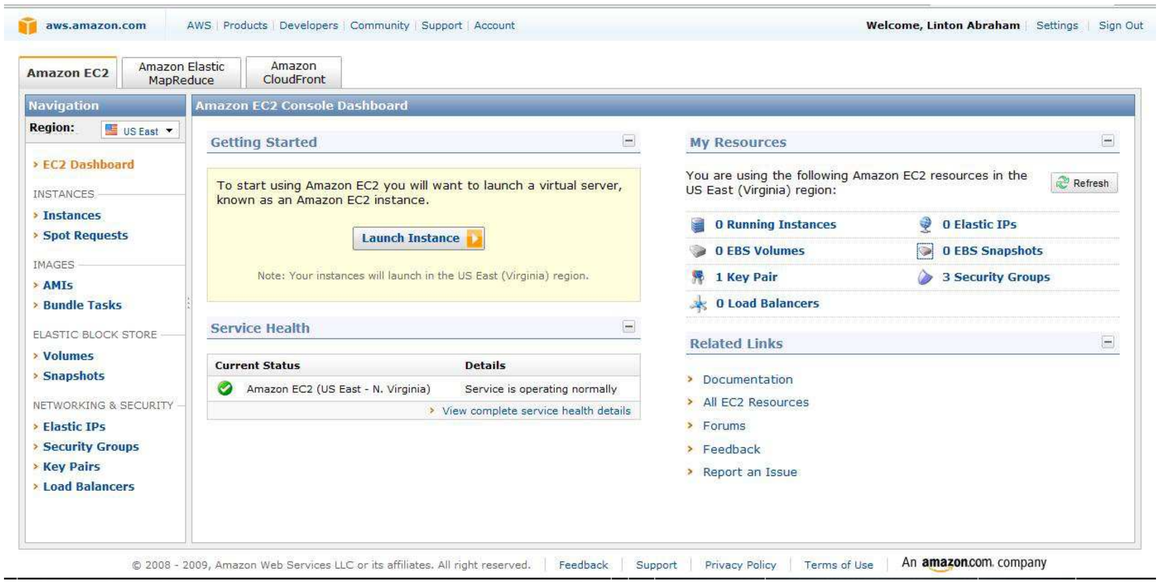
Figure 4.5: Amazon EC2 Primary Distributed Architecture[2]

4.3 VOC and the Amazon Elastic Compute Cloud(EC2)

4.3.1 Experiments

This section discusses a prototype implementation of a system for dynamically provisioning nodes across multiple grid sites. As illustrated in Figure 3.8, the user submits jobs to the VOs Grid Compute Element (CE). The user does not need to be concerned about how the resources he requires are obtained and allocated. The dynamic provisioning (resizing of the VOC) is performed by a watchdog that is set up on the VOC head node. In the current implementation, the watchdog uses a simple greedy algorithm. The watchdog monitors incoming jobs on the gatekeeper. When a job enters the local queue, the watchdog determines whether or not it should start a Virtual Machine (VM) on one of the physical hosts. If started, the virtual machine then joins the IPOP pool. The watchdog continues to monitor the job queue and starts VMs depending on the workload. The local scheduler (e.g. Condor [64], PBS [56], LSF [49]) schedules the jobs on the virtual machines. Once jobs are executed and there are more VMs than jobs, the watchdog terminates idle VMs.

Figure 3.8 also shows that each VO has one watchdog. Once all the local compute resources



To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the US East (Virginia) region.

Current Status	Details
✓ Amazon EC2 (US East - N. Virginia)	Service is operating normally

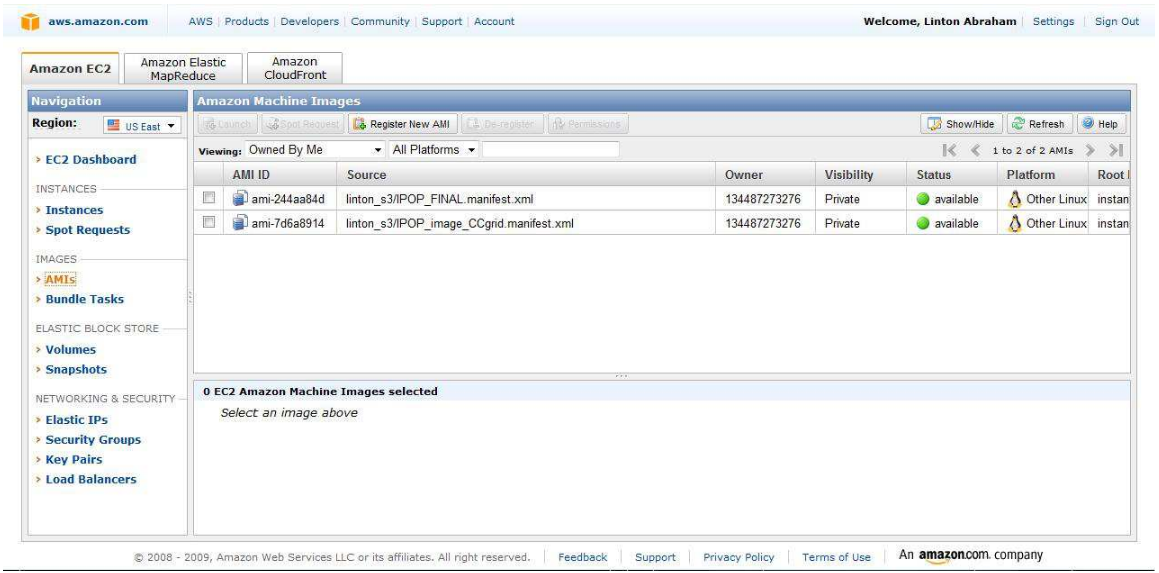
› View complete service health details

You are using the following Amazon EC2 resources in the US East (Virginia) region:

0 Running Instances	0 Elastic IPs
0 EBS Volumes	0 EBS Snapshots
1 Key Pair	3 Security Groups
0 Load Balancers	

© 2008 - 2009, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use | An amazon.com. company

Figure 4.6: Amazon provide a web UI as well as a command line interface(not in figure) for managing instances



	AMI ID	Source	Owner	Visibility	Status	Platform	Root
☐	ami-244aa84d	linton_s3/IPOP_FINAL.manifest.xml	134487273276	Private	● available	Other Linux	instan
☐	ami-7d6a8914	linton_s3/IPOP_image_CCgrid.manifest.xml	134487273276	Private	● available	Other Linux	instan

0 EC2 Amazon Machine Images selected

Select an image above

© 2008 - 2009, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use | An amazon.com. company

Figure 4.7: Amazon Console Management Instances page displaying the IPOP images that were used for experiments

are exhausted, the watchdog contacts the gatekeeper on another participating grid site. It then starts up VMs on that site and those machines join the VOs pool as well.

The IPOP bootstrap infrastructure used was that provided by PlanetLab. PlanetLab is a global research network that supports the development of new network services. More than 1,000 researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing[30]. UFL[1] has deployed more than 700 nodes that form a resilient IPOP bootstrap infrastructure that sever a Global community of scientific researchers and other users.

Experiments were conducted with a lease based system called the Amazon Elastic Compute Cloud (EC2) [24]. The local (on-site) compute resource consisted of 16 VMs dedicated to the Engage VO [11], which consisted of 16 dual core machines with each core dedicated to a VO(The setup was explained in detail in 4.1). The implemented watchdog is able to handle both machines in the local cluster and on Amazon EC2. The local cluster specifications and the original implementation of the watchdog are explained in detail in [54]. The watchdog for this experiment was designed to handle the complexities of multiple grid sites, the local cluster and a set of EC2 instances in this case.

The basic watchdog algorithm(designed to handle a single site) is:

1. Invoke PROBE_STATE, a module to determine the number of jobs in the schedule's queue,
2. Examine the jobs to VMs ratio and enter a the appropriate state, described below,
3. Take an action based on the current state,
4. Wait for a given interval (10 seconds in the prototype).

While the VOC Model supports any batch scheduler, the prototype implementation uses the Condor job manager [64]. The PROBE_STATE module is used to determine the state of the Condor queue when invoked. One of four possible states is then entered:

1. Number of Jobs in the queue is GREATER than the number of VMs reporting AND limit on number of local VMs HAS NOT been reached.
2. Number of Jobs in the queue is GREATER than the number of VMs reporting AND limit on number of local VMs HAS been reached.

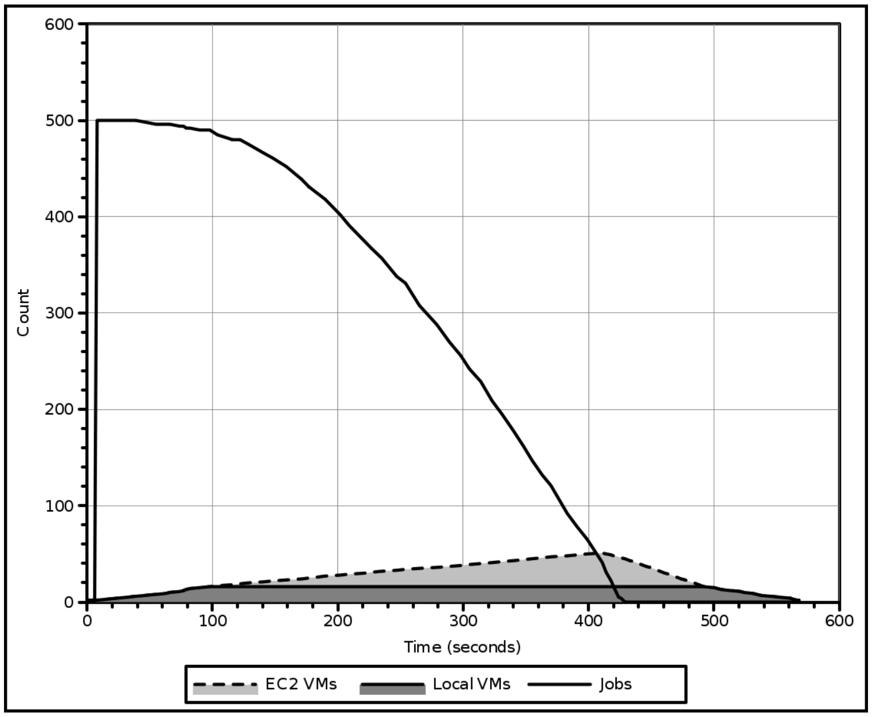


Figure 4.8: Short Jobs executing in Multiple Grid sites over IPOP

3. Number of Jobs in the queue is LESSER than the number of VMs reporting AND MORE VMs than the local cluster's limit are reporting.

4. Number of Jobs in the queue is LESSER than the number of VMs reporting AND LESSER VMs than the local cluster's limit are reporting.

Separate modules handle each state.

- State 1 invokes a START_LOCAL module that only starts machines on the local cluster. The module communicates with the gatekeeper using Globus. The gatekeeper contacts the head node to start up local VMs.

- State 2 invokes a START_EC2 module used to start EC2 instances, once local resources have been exhausted. The module uses the standard EC2 API command ec2-run-instances to boot an instance. Both the local and EC2 nodes are configured to join the IPOP pool and are part of a single VO. Each Amazon instance is pre-configured with the software packages that are required by the VO. The nodes are also configured to join the IPOP overlay once booted. Since Amazon bills a partial hour of usage as a full hour, a limit of a 100 instances is imposed

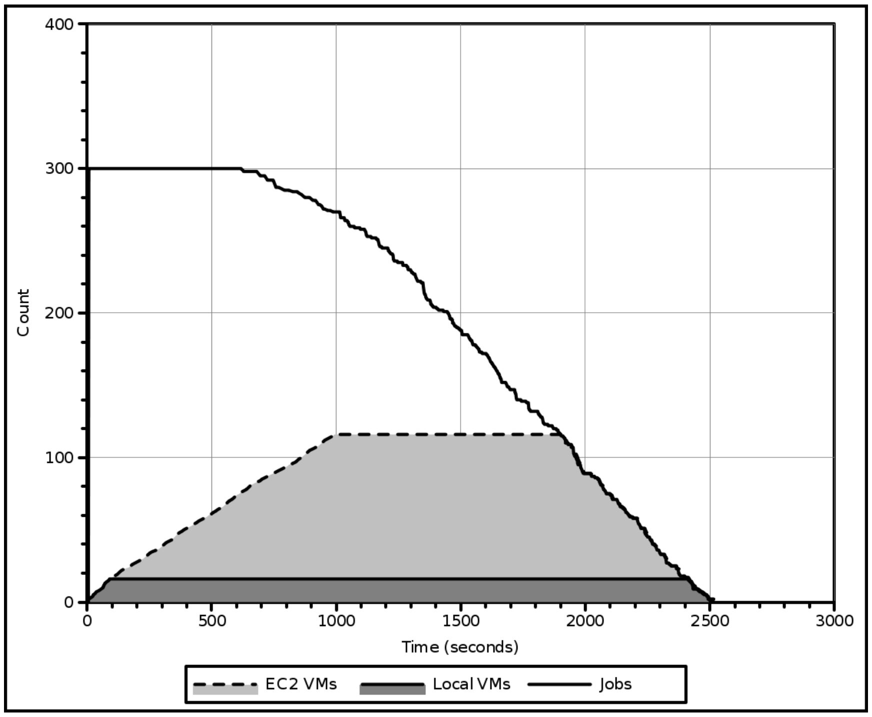


Figure 4.9: Long Jobs executing in Multiple Grid sites over IPOP

in order to control costs. Thus an EC2_instance_counter is maintained to ensure that the limit isn't exceeded.

- State 3 invokes a STOP_EC2 module when the number of jobs is less than the number of VMs. The STOP_EC2 module invokes another module called CHECK_IDLE. This module ensures that only the IP addresses of idle VMs are returned, guaranteeing that VMs that are scheduled and claimed will not be terminated. The system design ensures that all EC2 instances are terminated before the termination of the local VMs, ensuring that local resources are more heavily utilized than resources provided by off-site grid locations. Once all the instances are terminated, the watchdog shuts down VMs in the local cluster after ensuring that the number of jobs is less than the number of VMs

- State 4 invokes a STOP_LOCAL module that shuts down local VMs. As in the STOP_EC2 module, the CHECK_IDLE module returns idle VMs. Using this list, the STOP_LOCAL module terminates VMs that are idle. In both the STOP_LOCAL and STOP_EC2 modules, the systems are terminated by sending the 'poweroff' signal over secure shell.

4.3.2 Results

Figure 4.8 shows the short jobs that were run. Five hundred, 10 second sleep jobs submitted in one batch. As seen in the figure, since the jobs are short and embarrassingly parallel, the EC2 instances and local VMs are still running for a period of time after all the jobs are complete. In Figure 4.9, three hundred 10 second sleep jobs were submitted in one batch. As these were longer jobs, the instances were terminated before the all the jobs were completed because of the use of the local VMs. Therefore, the system is better suited to longer jobs due to the cost factor inherent in using the leased resources. The use of compute resources located at multiple grid sites ensures that adequate compute resources are always available to the VO and hence the end user. The overlay network allows nodes in any geographic location to be a part of the VO. To the user, it all appears as a large number of machines in the same pool.

Chapter 5

Conclusions

The use of an overlay system that was developed on a DHT-based P2P library proved to be scalable and easily configurable. IPOP proved to be the ideal networking medium for the VOC model, but does not limit VOCs by claiming to be an exclusive solution. The tests done on a single site(locally) was compared to the tests done without an overlay[54]. The overhead introduced by the overlay was negligent in comparison to that resulted by Globus and the Condor scheduler. The latency is also acceptable as the VOC model targets compute bound jobs. 4.2 showed how the utilization decreases as the jobs terminate quickly, lending an expected delay in the termination of nodes. 4.4, on the other hand, showed an improved system utilization and job execution time as ratio of jobs to VMs at any given time remained close to “1”. This also shows that the VOC model along with the Watchdog and an overlay network is better suited for large jobs that have large execution times(if operating on a single machine or tradition grid environment). The jobs used were embarrassingly parallel and synthetic(small and large batches of ’sleep’ jobs). However, this workload represents compute bound jobs well.

The tests conducted on multi-sites showed the scalability and resilience of the system. IPOP nodes were located in different geographic locations and were yet able to function efficiently as one Virtual cluster. The overheads introduced here is again negligible, for the same reasons stated above. Amazon limited the total number of instances to a ’100’. As the jobs came in, the nodes were booted up; at its peak giving a total of 116 compute nodes(16 on-site + 100 off-site). This is clearly shown in terms of short jobs in 4.8 and in terms of long jobs in 4.9. Various policies could be added to a site which would change administration but the system would essentially function the same.

Most Industrial establishments and educational institutions avoid Peer-to-peer based systems due to the immense network bandwidth that such systems require. There have also been copyright infringement issues associated p2p based file sharing utilities. But the use of p2p systems in a load-balancing environment where the use of DHTs speed up the lookup of information has many advantages that cannot be avoided. This thesis set out to answer a research question that revolved around using existing infrastructure in an efficient manner to provide compute resources continuously and cost-effectively based on work-load. The system designed here boasts of little to no user involvement in its functioning. As a matter of fact, once the user submits a job to his/her VO, he/she does not have to be concerned about how, where and when his resources are acquired from. All he does is watch his job queue for completion. And finally, with the acquiring of resources when needed, the system has very high utilization. Nodes that are not used for a defined period of time will be shut down, unless otherwise dictated by site policies. This ensures energy conservation and further cost effectiveness. All this is accomplished autonomously and dynamically.

For Future Work, adaptive and dynamic watchdog algorithms needs to be researched and developed. Also, the use of more efficient job schedulers that have less dependency needs to be used. This will also encourage the deployment of compute intensive and other high performance computing jobs on such widely-used and widely-available p2p clouds.

Bibliography

[1] Acis lab. Available from: http://www.acis.ufl.edu/.

[2] The amazon ec2 architecture. Available from: http://aws.typepad.com/.m/aws/2009/05/new-aws-load-balanci

[3] The clemson palmetto cluster. Available from: http://citi.clemson.edu/palm_arch.

[4] Cloud computing simplified. Available from: http://ivanov.wordpress.com/2008/05/01/cloud-computing/.

[5] Cloud computing: The evolution of a software as a service. Available from: http://knowledge.wpcarey.asu.edu/article.cfm?articleid=1614.

[6] Cloud computing wikipedia entry. Available from: http://en.wikipedia.org/wiki/Cloud_computing.

[7] Dht wikipedia entry. Available from: http://en.wikipedia.org/wiki/Distributed_hash_table.

[8] The differences between peer-to-peer systems and client-server systems. Available from: http://freepctech.com/pc/002/networks007.shtml.

[9] Distributed hash tables. Available from: http://www.linuxjournal.com/article/6797.

[10] Does parallel really mean fast? does fast really mean efficient? Available from: https://computing.llnl.gov/tutorials/parallel_comp/.

[11] Engage VO. Available from: https://twiki.grid.iu.edu/bin/view/Engagement/WebHome.

[12] Ibm m44 44x wikipedia entry. Available from: http://en.wikipedia.org/wiki/IBM_M44/44X.

[13] Introduction to distributed systems. Available from: http://code.google.com/edu/parallel/dsd-tutorial.html

[14] Kademlia: Wikipedia entry. Available from: http://en.wikipedia.org/wiki/Kademlia.

[15] More about ipop - acis ufl. Available from: http://socialvpn.wordpress.com/about/.

[16] The palmetto cluster at clemson. Available from: http://citi.clemson.edu/files/palmetto/breeze/palmettof

[17] Pastry dht. Available from: http://research.microsoft.com/en-us/um/people/antr/PASTRY/.

[18] Rfc for network address translation. Available from: http://www.faqs.org/rfcs/rfc1631.html.

[19] The rise of super computers. Available from: http://www.nap.edu/openbook.php?record_id=11148&page=157.

[20] Stun - simple traversal of udp through nat rfc. Available from: http://tools.ietf.org/html/rfc3489.

[21] Understanding high throughput computing. Available from: http://www.cs.wisc.edu/condor/htc.html.

[22] Virtual organizations and the open science grid. Available from: `http://www.opensciencegrid.org/About/Learn_About_Us/OSG_Organization/VOs`.

[23] Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, and Xiaomin Zhu. From virtualized resources to virtual computing grids: the In-VIGO system. *Future Generation Computer Systems*, 21(6):896–909, June 2005.

[24] Amazon Web Services. Amazon elastic compute cloud (amazon EC2). Available from: `http://aws.amazon.com/ec2/`.

[25] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[26] P. O. Boykin, J. S. A. Bridgewater, J. S. Kong, K. M. Lozev, B. A. Rezaei, and V. P. Roychowdhury. A symphony conducted by Brunet. Online, September 2007. Available from: `http://arxiv.org/abs/0709.4048`.

[27] Mark Burgess and Ricky Ralston. Distributed resource administration using Cfengine. *Software – Practice and Experience*, 27(9):1083–1101, 1997.

[28] Philip H. Carns, Walter B. Ligon, Robert B. Ross, and Rajeev Thakur. PVFS: A parallel file system for Linux clusters. In *ALS'00: Proceedings of the 4th annual Linux Showcase and Conference*, 2000.

[29] Jeffrey S. Chase, David E. Irwin, Laura E. Grit, Justin D. Moore, and Sara E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, June 2003.

[30] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003. `doi:http://doi.acm.org/10.1145/956993.956995`.

[31] Clemson University Cyberinfrastructure Research Group. Cirg home page. Available from: `http://cirg.cs.clemson.edu`.

[32] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research & Development*, 25(5):483–490, 1981.

[33] R. Davoli. VDE: Virtual Distributed Ethernet. In *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom 2005)*, Trento, Italy, February 2005.

[34] Miron Livny Douglas Thain, Todd Tannenbaum. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17:323–356, 2005. Available from: `http://dx.doi.org/10.1002/cpe.938`.

[35] Wesley Emeneker and Dan Stanzione. Dynamic virtual clustering. In *2007 IEEE International Conference on Cluster Computing*, 2007.

[36] Michael Fenn, Michael A. Murphy, and Sebastien Goasguen. A study of a KVM-based cluster for grid computing. In *47th ACM Southeast Conference (ACMSE '09)*, Clemson, SC, March 2009.

[37] Renato Figueiredo, Peter A. Dinda, and Jose Fortes. Resource virtualization renaissance. *Computer*, 38(5):28–31, May 2005.

[38] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A case for grid computing on virtual machines. In *23rd International Conference on Distributed Computing Systems*, 2003.

[39] Ian Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, February 2002.

[40] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[41] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, 15(3):200–222, 2001.

[42] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo. IP over P2P: Enabling self-configuring virtual IP networks for grid computing. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006. `doi:10.1109/IPDPS.2006.1639287`.

[43] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo. WOW: Self-organizing wide area overlay networks of virtual workstations. In *15th IEEE International Symposium on High Performance Distributed Computing*, 2006. `doi:10.1109/HPDC.2006.1652133`.

[44] A. Ganguly, D. Wolinsky, P. O. Boykin, and R. Figueiredo. Decentralized dynamic host configuration in Wide-area Overlays of virtual Workstations. In *IEEE International Parallel and Distributed Processing Symposium IPDPS 2007*, 2007. `doi:10.1109/IPDPS.2007.370664`.

[45] Laura Grit, David Irwin, Aydan Yumerefendi, and Jeff Chase. Virtual machine hosting for networked clusters: Building the foundations for 'autonomic' orchestration. In *First International Workshop on Virtualization Technology in Distributed Computing (VTDC '06)*, Tampa, FL, November 2006.

[46] David Irwin, Jeff Chase, Laura Grit, Aydan Yumerefendi, David Becker, and Ken Yocum. Sharing network resources with brokered leases. In *USENIX Technical Conference*, Boston, MA, June 2006.

[47] Xuxian Jiang and Dongyan Xu. VIOLIN: Virtual Internetworking on OverLay INfrastructure. Technical Report CSD TR 03-027, Purdue University, July 2003.

[48] Ivan Krsul, Arijit Ganguly, Jian Zhang, Jose A. B. Fortes, and Renato J. Figueiredo. VM-Plants: Providing and managing virtual machine execution environments for grid computing. In *ACM/IEEE Conference on Supercomputing*, 2004.

[49] Ian Lumb and Chris Smith. *Scheduling attributes and platform LSF*, chapter 12, pages 171–182. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[50] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: distributed hashing in a small world. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[51] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer LNCS, 2002. Available from: `http://www.springerlink.com/content/2ekx2a76ptwd24qt`, `doi:10.1007/3-540-45748-8_5`.

[52] Michael A. Murphy, Linton Abraham, Michael Fenn, and Sebastien Goasguen. Autonomic clouds on the grid. *Journal of Grid Computing*, 8(1):1–18, March 2010. `doi:10.1007/s10723-009-9142-3`.

[53] Michael A. Murphy, Michael Fenn, and Sebastien Goasguen. Virtual Organization Clusters. In *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009)*, Weimar, Germany, February 2009.

[54] Michael A. Murphy, Brandon Kagey, Michael Fenn, and Sebastien Goasguen. Dynamic provisioning of Virtual Organization Clusters. In *9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '09)*, Shanghai, China, May 2009.

[55] Hideo Nishimura, Naoya Maruyama, and Satoshi Matsuoka. Virtual clusters on the fly - fast, scalable, and flexible installation. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007)*, May 2007.

[56] Bill Nitzberg, Jennifer M. Schopf, and James Patton Jones. *PBS Pro: Grid computing and scheduling attributes*, chapter 13, pages 183–190. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[57] Open Science Grid. Open Science Grid. Available from: `http://www.opensciencegrid.org`.

[58] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[59] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick. The Open Science Grid: Status and architecture. In *International Conference on Computing in High Energy and Nuclear Physics (CHEP '07)*, 2007.

[60] Red Hat. Kernel-based Virtual Machine. Available from: `http://www.linux-kvm.org` [cited 10 March 2009].

[61] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer LNCS, 2001. Available from: `http://www.springerlink.com/content/404522p56nm85503`, `doi:10.1007/3-540-45518-3_18`.

[62] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM. `doi:http://doi.acm.org/10.1145/383059.383071`.

[63] Ananth I. Sundararaj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *Third Virtual Machine Research and Technology Symposium*, San Jose, CA, May 2004.

[64] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[65] M. Tsugawa and J. A. B. Fortes. A virtual network (ViNe) architecture for grid computing. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006. `doi:10.1109/IPDPS.2006.1639380`.

[66] David Isaac Wolinsky, Yonggang Liu, Pierre St. Juste, Girish Venkatasubramanian, and Renato Figueiredo. On the design of scalable, self-configuring virtual networks. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM. `doi:http://doi.acm.org/10.1145/1654059.1654073`.

[67] X. Zhang, K. Keahey, I. Foster, and T. Freeman. Virtual cluster workspaces for grid applications. Technical Report ANL/MCS-P1246-0405, Argonne National Laboratory, April 2005.