Automatic, Run-time and Dynamic Adaptation of Distributed Applications Executing in Virtual Environments

Thesis Proposal

Ananth I. Sundararaj
Department of Electrical Engineering and Computer Science
Northwestern University, Evanston, IL 60208-3118
ais@cs.northwestern.edu

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

October 20, 2005

Thesis Committee:

Peter A. Dinda (Advisor)
Assistant Professor

Fabián E. Bustamante — Assistant Professor
Aleksandar Kuzmanovic — Assistant Professor

Department of Electrical Engineering and Computer Science
Northwestern University, Evanston, IL 60208-3118

José A.B. Fortes (External committee member)
Professor, Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32611-6200

Abstract

Over the last fifteen years we have seen a tremendous increase in computer and network speeds and performance, resulting in the emergence of a new computing paradigm, wide-area distributed computing. However, the full potential of wide-area distributed computing has not been exploited, primarily due to the challenges involved in developing applications for such environments. The wide fluctuations in available resources and inherent heterogeneity of distributed environments require adaptation in each application. Such custom adaptation is exceedingly complex as the application requirements, computational and network resources can vary over time resulting in adaptation mechanisms and control not being common on today's applications.

We believe that one way of realizing the full potential of wide-area distributed computing is by building a virtual execution environment consisting of operating system level virtual machines connected by virtual networks. Virtualization technology such as Virtual Machine Monitors (VMMs) can greatly simplify distributed computing by lowering the level of abstraction from traditional units of work, such as jobs, processes, or RPC calls to that of a raw machine.

Such execution environments make possible low-level, application-, developer-, and user-independent adaptation mechanisms such as VM migration, overlay topology configuration and routing, network and CPU reservations. It is my thesis that an automatic adaptation control system guided by a single optimization scheme can exploit these mechanisms effectively for a broad range of distributed applications.

I have previously built and evaluated an Ethernet layer virtual network tool, VNET, that interconnects virtual machines and creates the illusion that they are on the user's local area network. I extended it with application independent adaptation mechanisms thus creating an adaptive virtual execution environment. Our group has previously shown how VNET is in an ideal position to monitor the application and the network. We have demonstrated that guided by the measured network resource information and inferred application demands, adaptation mechanisms controlled by greedy heuristics are effective in the context of two classes of distributed applications. I have also the laid the ground for formalizing the adaptation problem.

In my thesis, I will extend this work to show that adaptive virtual environments based on a single optimization scheme can be effective for a wide range of distributed applications. As a first step, I will study and port a wide range of distributed applications and benchmarks to my system and design a single optimization scheme that, I claim, will be effective for all of them. Next, building on my previous work on adaptation problem analysis I will formalize the complete adaptation problem and analyze its computational complexity. I will then design an adaptation algorithm(s) and implement it in my existing framework and close the loop by evaluating the effectiveness of my approach over a wide range of distributed applications.

This work has the potential to fill an important gap in distributed systems research by providing an automatic, run-time and dynamic adaptation scheme that is effective for a wide range of unmodified applications running on unmodified operating systems without requiring any developer or user help. It will help scientific and business computing in fully exploiting the powerful paradigm of wide-area distributed computing.

1. Introduction

Over the last few years we have seen a tremendous increase in computer and network performance. Despite this there are many applications in science, engineering, business and the arts, that cannot be effectively dealt with the current generation of co-located computer clusters and supercomputers. The popularity of the Internet coupled with the availability of low cost commodity computers and high-speed networks have enabled a shift in how computers are used. Computing resources distributed geographically and under different administrative domains can now be harnessed to provide a wide-area distributed computing environment, much more flexible and powerful than any single supercomputer. Variants of this new approach have been known by several names such as metacomputing, scalable computing, global computing, Internet computing, grid computing [31] or simply as wide-area distributed computing.

However, the full potential of wide-area distributed computing has not been exploited primarily due to the challenges involved in developing applications for such environments. The wide range of operating systems, specific versions of supporting software packages or software environment variations, in general, have made developing truly portable distributed applications a challenging task. Further, distributed environments are highly heterogeneous and experience wide fluctuations in available network and computational resources that require adaptation in each application to achieve even reasonable performance. Despite many efforts [89, 98, 54, 11], adaptation mechanisms and control are not common on today's applications. This is beacuse they tend to be both very application-specific and require considerable user or developer effort. Custom adaptation by either the user or the resource provider is exceedingly complex as the application requirements, computational and network resources can vary over time.

We believe that one solution to these problems is to adopt a new paradigm to wide-area distributed computing, namely, virtual machine distributed computing, leveraging operating system level virtual machine (VM) technology. This technology dates back to the 1970s when IBM achieved commercial success by introducing virtualization technology thus allowing its mainframes to be used in a time shared manner. Over the next decade interest in hardware virtualization declined partly due to the introduction of affordable low end computers. Despite this the general notion of resource virtualization has existed throughout, addressing computer systems problems by adding an additional layer of abstraction. Over the past few years we have seen a resurgence of interest in hardware virtualization technology in the form of commercial products from VMware [95] and IBM [45] and increased activity in the research community [27]. The virtual machine technology relevant to my work is discussed in detail in Section 12.2. In brief, my work builds on operating-system level virtual machines, more specifically Virtual Machine Monitors, such as VMware [95] and IBM's VM [45]. These present an abstraction identical to a physical machine. For example, VMware provides the abstraction of an Intel IA32-based PC (including memory, IDE or SCSI disk controllers, disks, network interface cards, video card, BIOS, etc.) On top of this abstraction, almost any existing PC operating system environment can be installed and run. The overhead of this emulation has been shown to be quite low [83, 28].

Virtualization technology such as VMMs can greatly simplify wide-area distributed computing by low-

ering the level of abstraction from traditional units of work, such as jobs, processes, or RPC calls to that of a raw machine. This abstraction makes resource management easier from the perspective of resource providers and results in lower complexity and greater flexibility for resource users. A virtual machine image that includes preinstalled versions of the correct operating system, libraries, middleware and applications can make the deployment of new software far simpler. The first detailed case for distributed computing on virtual machines appeared in a previous paper [28]. We have been developing a middleware system, Virtuoso, for virtual machine distributed computing [78]. Others have shown how to incorporate virtual machines into the emerging grid standards environment [53].

Distributed computing is intrinsically about using multiple sites, with different network management and security philosophies, often spread over the wide area [31]. Running a virtual machine on a remote site is equivalent to visiting the site and connecting a new machine. The nature of the network presence (active Ethernet port, traffic not blocked, routable IP address, forwarding of its packets through firewalls, etc.) the machine gets, or whether it gets a presence at all, depends completely on the policy of the site. Not all connections between machines are possible and not all paths through the network are free. The impact of this variation is further increased as the number of sites is increased and if we permit virtual machines to migrate from site to site.

To deal with this network management problem in Virtuoso, I developed and implemented VNET [86], a simple link layer virtual network tool. Using VNET, virtual machines have no network presence on a remote site. Instead, VNET provides a mechanism to project their virtual network cards onto another network, which also moves the network management problem from a remote network to another network of the user's choosing. Because the virtual network is a link layer network, a machine can be migrated from site to site without changing its presence—it always keeps the same IP address, routes, etc. VNET is publicly available.

This provides a distributed application developer with the abstraction of multiple physical machines all connected to the user's network that can be easily managed, while in reality they are virtual machines, hosted in different geographic locations under different administrative domains. Virtual networks are not just a required means for providing network connectivity to virtual machines, they have tremendous potential to become adaptive overlay networks. I extended VNET to become an adaptive overlay network by building into it application independent adaptation mechanisms. I have previously shown how adaptive virtual networks can perform automatic services on behalf of the application without requiring any changes in the applications themselves. I believe that such a system design is central to fully exploiting the potential of wide-area distributed computing.

Our group has previously shown how VNET is in an ideal position to

1. Measure the traffic load and application topology of the virtual machines [37], accomplished using the Virtual Topology and Traffic Inference Framework, VTTIF[1], now integrated with VNET.

[1] VTTIF was designed and implemented by my fellow graduate student, Ashish Gupta.

2. Monitor the underlying network and its topology [38], accomplished using the Wren[2] (Watching Resources from the Edge of the Network) tool, integrated with VNET.

3. Adapt the application as measured in step 1 to the network as measured in step 2[3] [85, 87], and

4. Adapt the network to the application by taking advantage of resource reservation mechanisms[4] [59].

These services can be performed on behalf of *existing, unmodified applications and operating systems* running in the virtual machines.

Thesis statement: An emerging abstraction is to perform distributed computing inside of virtual machines (VMs) interconnected with a virtual network. Such execution environments make possible low-level, application-, developer-, and user-independent adaptation mechanisms such as VM migration, overlay topology configuration and routing, network and CPU reservations. It is my thesis that an automatic adaptation control system guided by a single optimization scheme can exploit these mechanisms effectively for a broad range of distributed applications.

In my thesis, I will extend our previous work to show that adaptive virtual environments based on a single optimization scheme can be effective for a wide range of distributed applications. As a first step, I will study and port a wide range of distributed applications and benchmarks to my system and design a single optimization scheme that, I claim, will be effective for all of them. Next, building on my previous work on adaptation problem analysis and based on the proposed single optimization scheme, I will formalize the complete adaptation problem and analyze its computational complexity. I will then design an adaptation algorithm that guided by the inferred application demands (using VTTIF) and measured network resources (using Wren or a similar tool), drive the adaptation mechanisms (VNET topology, routing, VM migration, network reservation). Finally I will implement the algorithm in my existing working framework and close the loop by evaluating the effectiveness of my approach over a wide range of ported distributed applications.

This work is an important step in realizing the full potential of grid computing and in general, wide-area distributed computing. Such an environment will ease the burden of adaptation on distributed application developers, thus helping scientific and business computing to fully exploit the powerful paradigm of wide-area distributed computing.

We begin this proposal in Section 2 with a description of the Virtuoso system, our middleware for virtual machine distributed computing. The adaptive virtual networking component of Virtuoso, VNET, is described in Section 3. The application resource demand inference using VTTIF is discussed in Section 4 and Section 5 discusses physical network measurements in the context of this thesis. Virtuoso's application-independent

[2]Wren was designed and implemented by our collaborators, Marcia Zangrilli and Bruce B. Lowekamp at the Department of Computer Science, College of William and Mary, Williamsburg, VA.

[3]This was joint work with my fellow graduate student Ashish Gupta and our advisor, Peter A. Dinda.

[4]This was joint work with my fellow graduate student John R. Lange and our advisor Peter A. Dinda.

adaptation mechanisms are described in Section 6. We lay the groundwork for the formulation of the adaptation problem, analysis of its computational complexity and possible approximate solutions in Section 7. Section 8 illustrates a possible problem formulation based on a specific metric. Some preliminary results are presented in Section 9. Section 10 outlines the thesis work and Section 11 describes its main contributions. Finally Section 12 compares and contrasts this work with related literature.

2. Virtuoso

We are developing middleware, Virtuoso, for virtual machine distributed computing that for a user very closely emulates the existing process of buying, configuring, and using an Intel-based computer or collection of computers from a web site, a process with which many users and certainly all system administrators are familiar.

In our model, the user visits a web site, much like the web site of Dell or IBM or any other company that sells Intel-based computers. The site allows him to specify the hardware and software configuration of a computer and its performance requirements, and then order one or more of them. The user receives a reference to the virtual machine which they can then use to start, stop, reset, and clone the machine. The system presents the illusion that the virtual machine is right next to the user, in terms of console display, devices, and the network. The console display is sent back to the user's machine, the CD-ROM is proxied to the user's machine's CD-ROM, and the virtual machine appears to be plugged into the network side-by-side with the user's machine. The user can then install additional software, including operating systems. The system is permitted to move the virtual machine from site to site to optimize its performance or cost, but must preserve the illusion. More details about the current Virtuoso implementation are available elsewhere [78].

We use VMware GSX Server [95] running on Linux as our virtual machine monitor. Although GSX provides a fast remote console, we use VNC [71] in order to remain independent of the underlying virtual machine monitor. We proxy CD-ROM devices using Linux's network block device, or by using CD image files. Network proxying is done using VNET, as described in the next section. It should be noted that VMware is not a requirement of either Virtuoso or any of the work proposed in this thesis.

3. VNET

As mentioned previously in Section 1, we are faced with an interesting network management problem in Virtuoso. The virtual machines purchased by an user through Virtuoso, can be hosted on multiple sites, with different network management and security philosophies, often spread over the wide area. Running a virtual machine on a remote site is equivalent to visiting the site and connecting a new machine. The network presence the virtual machine gets is completely dependent on the foreign site. This situation is further complicated as the number of sites is increased and if we permit virtual machines to migrate from site to site.

VNET [86] is the part of our system that creates and maintains the networking illusion that the user's virtual machines are on the user's local area network. It is a simple link layer proxying scheme that works

entirely at user level. Using VNET, virtual machines have no network presence at all on a remote site. Instead, VNET provides a mechanism to project their virtual network cards onto another network (the user's home network), which also moves the network management problem from one network to another. Because the virtual network is a link layer network, a machine can be migrated from site to site without changing its presence—it always keeps the same IP address, routes, etc.

The primary dependence it has on the virtual machine monitor is that there must be a mechanism to extract the raw Ethernet packets sent by the virtual network card, and a mechanism to inject raw Ethernet packets into the virtual card. The specific mechanisms we use are packet filters, packet sockets, and VMWare's [95] host-only networking interface. VMWare, in its Workstation and GSX Server variants, can connect the virtual network interface to the network in three different ways. To the operating system running in the virtual machine (the VM), they all look the same. By themselves, these connection types are not well suited for use in a wide-area, multi-site environment that permit virtual machine migration from site to site, as described previously [86]. VMware's host-only networking scheme has the minimum possible interaction with network administration in the local environment. Here, a virtual interface is created on the host which is connected to the virtual interface in the VM. When brought up with the appropriate private IP addresses and routes, this enables programs on the host to talk to programs on the VM. Hence we use it as a mechanism to extract the raw Ethernet packets sent by the virtual network card.

Though we use VMware, specifically, VMWare GSX Server 2.5, as our Virtual Machine Monitor, VNET, without modification, has been successfully used with User Mode Linux [20] and the VServer extension to Linux [61].

Each physical machine that can instantiate virtual machines (a host) runs a single VNET daemon. One machine on the user's network also runs a VNET daemon. This machine is referred to as the Proxy. Figure 1 shows a typical startup configuration of VNET for four hosts, each of which may support multiple VMs. Each of the VNET daemons running on the foreign hosts is connected by a TCP connection (a VNET link) to the VNET daemon running on the Proxy. We refer to this as the resilient star backbone centered on the Proxy. By resilient, we mean it will always be possible to at least make these connections and re-establish them on failure.

3.1 Operation

The VNET daemons running on the hosts and Proxy open their virtual interfaces in promiscuous mode using Berkeley packet filters [66]. Each VNET daemon has a forwarding table, Figure 2 shows one such forwarding table at a VNET daemon. Each packet captured from the interface or received on a TCP connection is matched against this forwarding table to determine where to send it, the possible choices being sending it over one of its outgoing links (TCP / UDP) or writing it out to one of its local interfaces using libnet, which is built on packet sockets, available on both Unix and Windows. If the packet does not match any rule then no action is taken. For each packet multiple rules might be matched at the same time. Each match has a priority value associated with it, calculated dynamically based on the strength of the match. The stronger the

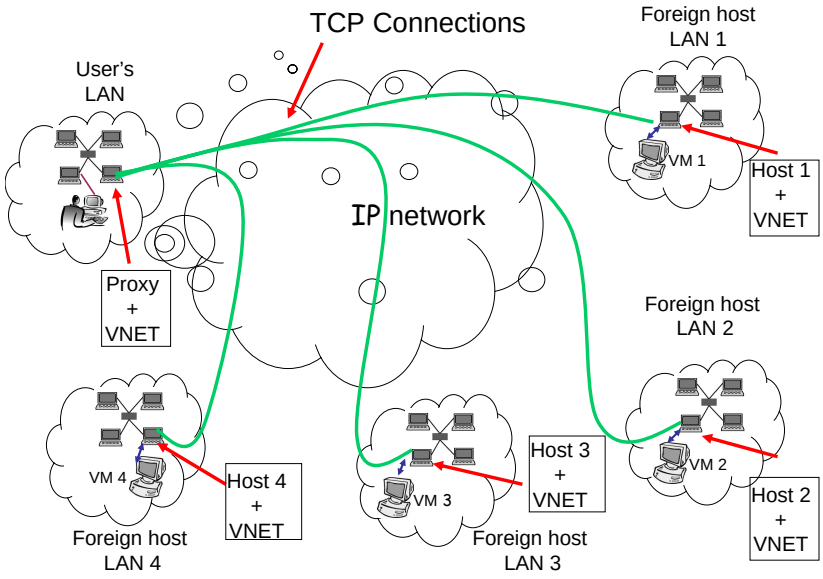

Figure 1. VNET startup topology.

Source Qualifier	Source Address	Destination Qualifier	Destination Address	Hop Start	Hop End	Interface
not	00:50:56:00:21:01	-	FF:FF:FF:FF:FF:FF	-	-	vmnet1
-	any	-	00:50:56:00:21:01	-	-	vmnet1
-	00:50:56:00:21:01	-	none	Host1	Proxy	-

Source Qualifier : "not" or "-"
Destination Qualifier : "not" or "-"

Source Address : Any valid Ethernet address
Destination Address : Any valid Ethernet address or "None"

Hop Start and Hop End : Physical machines that run VNET daemons
Hop Start – Hop End : TCP connection between those machines

Interface : Interface to which packet has to be injected

For any Ethernet packet multiple rules might be matched at the same time, each match has a priority value and the rule with the highest priority is used.

The rule that has the destination address as "none" is the default rule. This rule is always matched as long as the source address matches, but has the lowest possible priority. The packet would be sent over the TCP connection to the VNET daemon on the Proxy.

Figure 2. Portion of a routing table stored on the VNET daemon on a host.

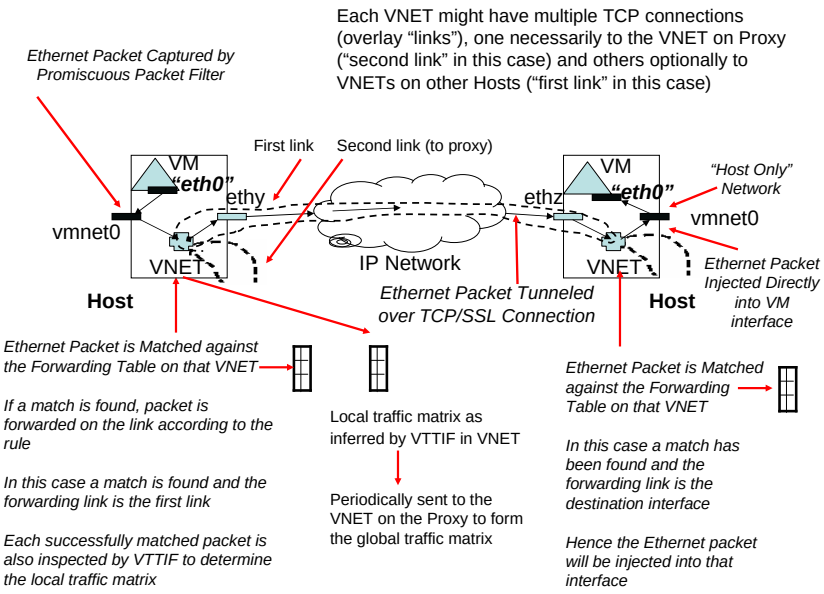

Figure 3. A VNET link.

match, the higher the priority value. For example, a rule matched with the "any" qualifier will have a lower priority than a rule matched with exact values. The rule with the highest priority is used. The rule that has the destination address as "none" is the default rule. This rule is always matched as long as the source addresses match, but has the lowest possible priority. If only the default rule is matched then the packet would be sent over the TCP connection to the VNET daemon on the Proxy.

Figure 3 helps to illustrate the operation of a VNET link. Each successfully matched packet is also passed to VTTIF to determine the local traffic matrix. Each VNET daemon periodically sends its inferred local traffic matrix to the VNET daemon on the Proxy. The Proxy, through its physical interface, provides a network presence for all the VMs on the user's LAN and makes their configuration a responsibility of the user.

The first generation of VNET was limited solely to this star topology [86], thus all traffic among the users' VMs would be forwarded through the central Proxy, resulting in extra latency and a bandwidth bottleneck. The star would be used regardless of the application, as its sole goal was to provide connectivity for the VMs regardless of the security constraints on the various sites.

The second generation VNET removes this restriction. Now, the star topology is simply the initial configuration, again to provide connectivity for the VMs. Additional links and forwarding rules can be added or removed at any time. This makes topology adaptation, as we describe in this thesis, possible. Figure 4 shows a VNET configuration that has been dynamically adapted. Also the links can either use TCP or UDP.

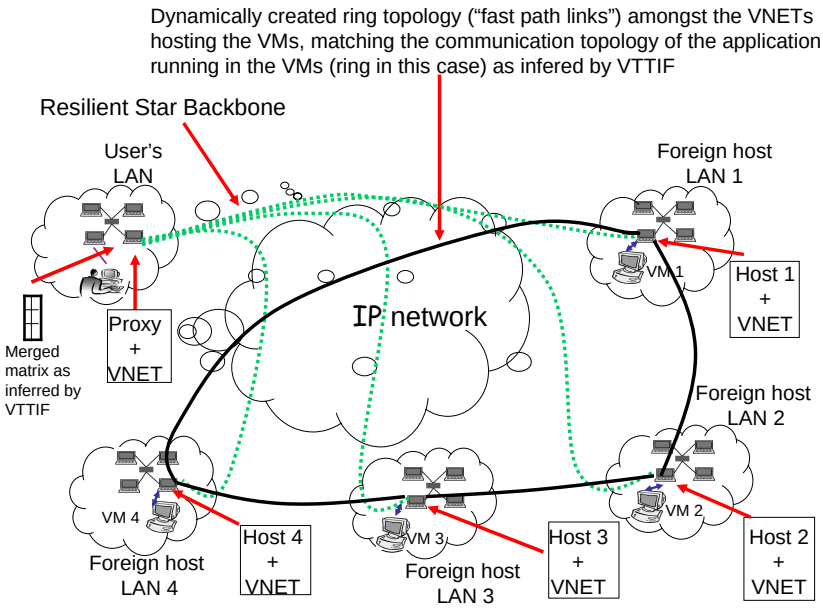

Figure 4. As the application progresses VNET adapts its overlay topology to match that of the application communication as inferred by VTTIF leading to an significant improvement in application performance, without any participation from the user.

3.2 VNET primitives

A VNET client can connect to any of the VNET daemons to query status or perform an action. Following are the primitives made available by VNET.

- Add an overlay link between two VNET daemons.
- Delete an overlay link.
- Add a rule to the forwarding table at a VNET daemon.
- Delete a forwarding rule.
- List the available network interfaces.
- List all the links to and from a VNET daemon.
- List all the forwarding rules at a VNET daemon.
- Set an upper bound on VNET configuration time.

The primitives generally execute in about 20 ms, including client time. On initial startup VNET can calculate an upper bound on the time taken to configure itself (or change topology). The last primitive is

a means of automatically passing this value to VTTIF, to be used to determine its sampling and smoothing intervals.

3.3 A language and its tools

Building on the primitives, we have developed a language for describing the topology and forwarding rules. Figure 5 defines the grammar for the language. The tools we use here take the form of scripts that generate or parse descriptions in that language. These tools provide functionality such as:

- Start up a collection of VNET daemons and establish an initial topology among them.
- Fetch and display the current topology.
- Fetch and display the route a packet will take between two Ethernet addresses.
- Compute the differences between the current topology with routing rules and a specified topology with routing rules.
- Reconfigure the topology and routing rules to match a specified topology and routing rules.
- Fetch and display the current application topology using VTTIF (described in Section 4).

3.4 VNET performance

The overhead of using VNET is mitigated over the wide area. In a 100 Mbps Ethernet LAN, VNET is as fast as state of the art network virtualization software [95], which in turn is almost as fast as the native hardware. However, over faster networks, such as optical networks operating at Giga-bit speeds or higher [47], VNET is a factor of two slower than commercial virtual network solutions [95], which in turn are a factor of 3 slower than the native hardware. We are currently working on improving the performance of VNET on faster networks. Below, we first describe the improvements engineered in the second generation VNET and then state our future plans for optimizing VNET performance.

3.4.1 UDP overlay links

In the first version of VNET, all VNET overlay links were TCP connections. The primary reason was to make it straightforward to support optional SSL encryption. This is not essential, since a VNET link is a virtual Ethernet layer link and thus needs to provide no guarantees of delivery, ordering, or corruption.

Running VNET over TCP results in lower than necessary throughput for applications running inside the VMs. Interaction between the TCP connection at the application layer and the TCP connection used for the VNET overlay dramatically reduces performance. A packet loss in the underlying VNET TCP connection will lead to a retransmission and hence a delay for the application's TCP connection, which in turn could time out and retransmit itself. The application's TCP connection will always detect a packet loss by the

$$
\begin{aligned}
\langle \text{program} \rangle &\longrightarrow BEGIN\ \langle \text{host} \rangle\ \langle \text{config} \rangle\ END \\
\langle \text{host} \rangle &\longrightarrow \langle \text{host} \rangle\ HOST\ \langle \text{username} \rangle\ AT\ \langle \text{machine} \rangle\ \langle \text{port} \rangle\ \langle \text{interface} \rangle \\
&\quad \mid \varepsilon \\
\langle \text{config} \rangle &\longrightarrow \langle \text{config} \rangle\ \langle \text{action} \rangle\ \langle \text{link} \rangle\ \langle \text{rules} \rangle \mid \varepsilon \\
\langle \text{action} \rangle &\longrightarrow ADD \mid DELETE \\
\langle \text{link} \rangle &\longrightarrow \langle \text{link} \rangle\ LINK\ \langle \text{link-type} \rangle\ \langle \text{machine} \rangle\ \langle \text{machine} \rangle\ \langle \text{machine} \rangle \\
&\quad \mid \varepsilon \\
\langle \text{link-type} \rangle &\longrightarrow TCP \mid UDP \\
\langle \text{rules} \rangle &\longrightarrow \langle \text{rules} \rangle\ FORWARD\ \langle \text{rule-type} \rangle\ \langle \text{machine} \rangle\ \langle \text{qualifier} \rangle \\
&\quad \langle \text{macaddress} \rangle\ \langle \text{qualifier} \rangle\ \langle \text{macaddress} \rangle\ \langle \text{destination} \rangle \\
&\quad \mid \varepsilon \\
\langle \text{rule-type} \rangle &\longrightarrow EDGE \mid INTERFACE \\
\langle \text{destination} \rangle &\longrightarrow \langle \text{machine} \rangle\ \langle \text{machine} \rangle \mid \langle \text{interface} \rangle \\
\langle \text{qualifier} \rangle &\longrightarrow NOT \mid ANY \mid \varepsilon \\
\langle \text{macaddress} \rangle &\longrightarrow \textit{Ethernet address} \\
&\quad \textit{such as } 01 : 02 : 03 : 04 : 05 : 06 \\
\langle \text{machine} \rangle &\longrightarrow \textit{Machine name} \\
&\quad \textit{such as machine1} \\
\langle \text{username} \rangle &\longrightarrow \textit{User account on machine} \\
\langle \text{port} \rangle &\longrightarrow \textit{Port where VNET daemon runs} \\
\langle \text{interface} \rangle &\longrightarrow \textit{Ethernet interface} \\
&\quad \textit{such as eth0} \\
\langle \text{at} \rangle &\longrightarrow AT
\end{aligned}
$$

Figure 5. Grammar defining the language for describing VNET topology and forwarding rules.

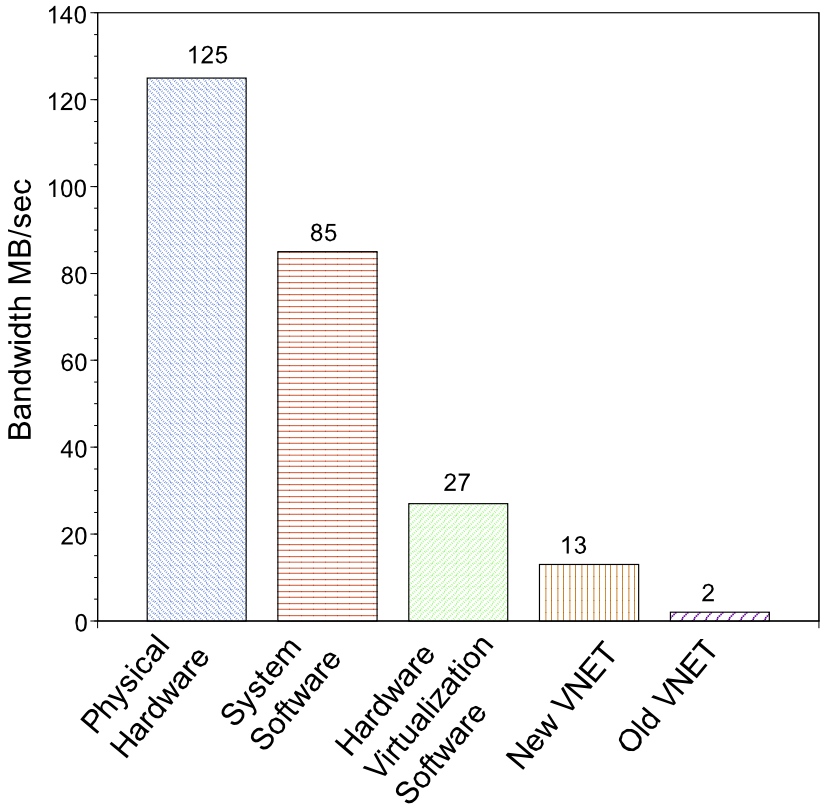

Figure 6. Throughput on a gigabit switch with different bottlenecks.

expiration of the retransmission timer rather than by receipt of triple duplicate acknowledgments. This will then always trigger slow start instead of fast retransmit, leading to reduced throughput.

VNET now supports creation of overlay links using UDP in addition to TCP. This increases the throughput seen by application-level TCP by a factor of two.

3.4.2 Improved forwarding rule lookup

VNET forwards Ethernet packets. When a packet arrives, it must look up the appropriate forwarding rule based on the packet's destination address. We have improved the lookup mechanism through a forwarding rule cache that gives us constant time lookups on average. This improved performance by a factor of three.

3.4.3 Further improving VNET performance

We have improved the performance of VNET, as measured on a dedicated gigabit Ethernet network, by a factor of six. However, there is still considerable room for improvement and we are a long way from being able to support gigabit speeds in VNET. Figure 6 illustrates where we stand.

Although not central to my thesis, I, in a combined effort with John R. Lange, intend to improve VNET

performance further by utilizing the memory-mapped I/O support in pcap to deliver more data from the VM to VNET for each context switch. Some additional possible extensions are

- To move the forwarding core of VNET into the Linux kernel on the host to avoid context switches in their entirety.
- To write a device driver for use inside the VM that will more efficiently deliver data to VNET.

4. VTTIF

The Virtual Topology and Traffic Inference Framework (VTTIF) enables topology inference and traffic characterization for applications running inside the VMs in the Virtuoso system. VTTIF infers the application's communication demands which along with the underlying network measurements guide the adaptation mechanisms. This component has been developed and implemented by my fellow graduate student Ashish Gupta [37]. As described earlier, such inference is important for automated adaptation where the underlying network and computational resources can be automatically adapted to the application's needs. VNET is ideally placed to monitor the resource demands of the VMs. In earlier work [37], Gupta et al. have demonstrated that it is possible to successfully infer the topology and traffic load matrix of a bulk synchronous parallel application running in a virtual machine-based distributed computing environment by observing the low level traffic sent and received by each VM. VTTIF is integrated with VNET.

4.1 Operation

VTTIF works by examining each Ethernet packet that a VNET daemon receives from or delivers to a local VM. VNET daemons collectively aggregate this information producing a global traffic matrix for all the VMs in the system, from which the topology can then be recovered by applying a simple normalization and pruning algorithm.

Gupta et al. were able to accurately recover many common topologies from both synthetic and application benchmarks like the PVM-NAS benchmarks. VTTIF adds very little overhead to VNET. Latency is essentially indistinguishable while throughput is effected on the order of 1%.

4.2 Continuous measurement and inference

VTTIF runs continuously, updating its view of the topology and traffic load matrix among a collection of Ethernet addresses being supported by VNET. Natural questions arise: How fast can VTTIF react to topology change? If the topology is changing faster than what VTTIF can react to, will it oscillate or provide a damped view of the different topologies? How sensitive is VTTIF to the choice of parameters?

VTTIF is configured by three parameters:

- Update rate: The rate at which local traffic matrix updates are sent from the VNET daemons to the VNET daemon running on the Proxy.

- Smoothing interval: The window over which the global traffic matrix on the Proxy is aggregated from the updates received from the other VNET daemons. This provides a low-passed view of the application's behavior.
- Detection threshold: The fraction of traffic intensity on the highest intensity link that must be present on any other link before it is considered to be a part of the topology.

The reaction time of VTTIF depends on the rate of updates from the individual VNET daemons. At fastest, these updates arrive at a rate of 20 Hz. Whether VTTIF reacts to an update by declaring that the topology has changed depends on the smoothing interval and the detection threshold. After VTTIF determines that a topology has changed, it will take some time for it to settle, showing no further topology changes. The best case settle time that Gupta et al. measured is one second. In other words, it can take as little as one second for VTTIF to discover a new topology.

Given some configured update rate, smoothing interval, and detection threshold, there is a maximum rate of topology change that VTTIF can keep up with. Beyond this rate, VTTIF has been designed to stop reacting, settling into a topology that is effectively a union of all the topologies that are unfolding in the network. Within limits, VTTIF is largely insensitive to the choice of detection threshold. However, this parameter does determine the extent to which similar topologies can be distinguished.

5. Network Measurements

To successfully build an adaptive system, in addition to inferring the application communication demands, we need to be able to measure the underlying network. The virtual overlay network can then be adapted in the most efficient and productive way. There is abundant work that suggests that underlying network measurements can be accomplished within or without the virtual network using both active [75, 100] and passive techniques [102, 64, 76]. In joint work with Ashish Gupta, Marcia Zangrilli, Bruce B. Lowekamp and Peter A. Dinda, I have shown that the naturally occurring traffic of an existing, unmodified application running in VMs can be used to measure the underlying physical network [38]. In this Section, I will briefly describe this work. However, at this point in time, I have not decided whether I will use Wren with VNET or some other third party network measurement tool. The main drawback with Wren is that it requires kernel modification.

5.1 Wren

Network measurement using Wren in VNET is based on Zangrilli et al.'s Wren passive monitoring and network characterization system [103, 102]. Many applications adapt to network performance simply by observing the throughput of their own network connections. VNET's natural abstraction of the underlying network makes such application-level adaptation more difficult because the application cannot accurately determine which network resources are in use. However, VNET is in a good position to observe an application's traffic itself. Because VNET does not alter that traffic, it can only observe the amount of traffic

naturally generated by the application. Because we are targeting applications with potentially bursty and irregular communication patterns, many applications will not generate enough traffic to saturate the network and provide useful information on the current bandwidth achievable on the network.

Watching Resources from the Edge of the Network (Wren) is designed to passively monitor applications' network traffic and use those observations to determine the available bandwidth along the network paths used by the application. The key observation behind Wren is that even when the application is not saturating the network it is sending bursts of traffic that can be used to measure the available bandwidth of the network. A good example of such an application is a typical scientific computing BSP-style application that sends short messages at regular intervals. Even though the application is not using all of the available bandwidth, we can determine the available bandwidth along that path and use that information to guide adaptation.

Wren consists of a kernel extension and a user-level daemon. Wren can:

- Observe every incoming and outgoing packet arrivals in the system with low overhead.
- Analyze these arrivals using state-of-the-art techniques to derive from them latency and bandwidth information for all hosts that the present host communicates with.
- Collect latency, available bandwidth, and throughput information so that an adaptation algorithm can have a bird's eye view of the physical network, just as it has a bird's eye view of the application topology via VTTIF.
- Answer queries about the bandwidth and latency between any pair of machines in the virtual network.

5.2 Wren and Virtuoso

Virtuoso and Wren are integrated by incorporating the Wren extensions into the Host operating system of the machines running VNET [38]. In this position, Wren monitors the traffic between VNET daemons, not between individual VMs. Both the VMs and VNET are oblivious to this monitoring, except for a small performance degradation. Figure 7 shows Virtuoso's interaction with Wren.

6. VADAPT: Adaptation in Virtuoso

Virtuoso uses VTTIF to determine the communication behavior of the application running in a collection of VMs and Wren to determine the behavior of the underlying network. Finally adaptation algorithms drive application independent adaptation mechanisms to automatically adapt unmodified applications at run-time to available resources thereby optimizing the application's performance. VADAPT is the component of Virtuoso that performs adaptation. Figure 7 shows the different components of Virtuoso salient to my thesis and how they interact to perform automatic, dynamic and run-time adaptation.

6.1 Adaptation mechanisms

We use the following adaptation mechanisms:

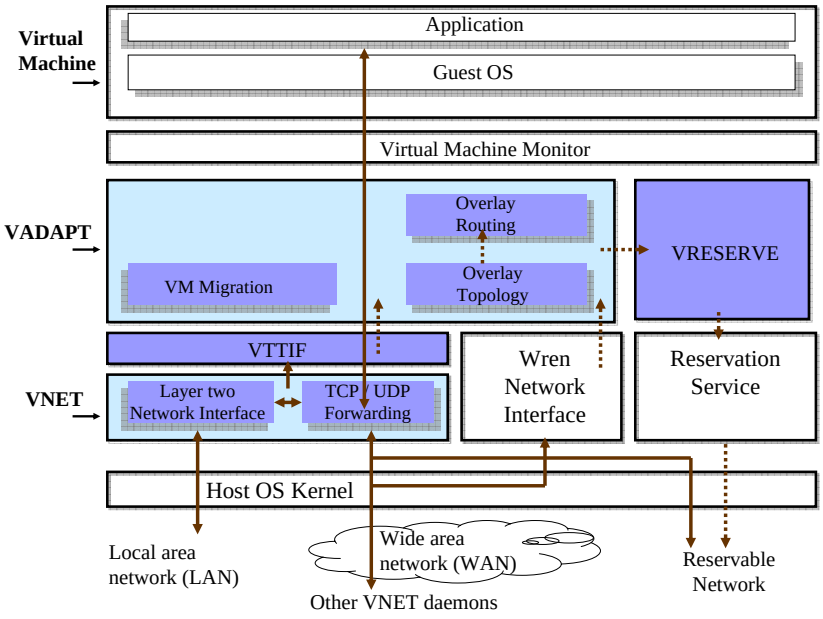

Figure 7. System overview. Dashed lines indicate control traffic, solid lines denote actual network traffic. The highlighted boxes are components of Virtuoso.

- **Virtual machine migration:** Virtuoso allows us to migrate a VM from one physical host to another. Much work exists that demonstrates that fast migration of VMs running commodity applications and operating systems is possible [70, 73, 57]. Migration times down to 5 seconds have been reported [57]. As migration times decrease, the rate of adaptation we can support and my work's relevance increases. Note that while process migration and remote execution has a long history [82, 23, 68, 91, 101], to use these facilities, we must modify or relink the application and/or use a particular OS. Neither is the case with VM migration.

- **Overlay topology modification:** VNET allows us to modify the overlay topology among a user's VMs at will. A key difference between it and overlay work in the application layer multicast community [7, 10, 44] is that the VNET provides global control of the topology, which our adaptation algorithms currently (but not necessarily) assume.

- **Overlay forwarding:** VNET allows us to modify how messages are routed on the overlay. Forwarding tables are globally controlled, topology and routing are completely separated, unlike in multicast systems.

- **Automatic dynamic run-time network reservations:** The VRESERVE [59] component of Virtuoso provides for automatic network reservations in networks that support reservations of bandwidth along paths. For example, various optical networks [47] support light-path set up, essentially allowing for an arbitrary topology to be configured. Currently, manual intervention is required to determine what

circuits are needed and how to provision them. When VNET operates over networks that support reservations, VRESERVE automatically determines the necessary paths and reserves them appropriately. Further, we can do so dynamically, changing paths and reservations at run-time as the communication needs of the applications change. VRESERVE alleviates the reservation responsibility for both the user and the developer. In fact the environment experienced by both is exactly the same as when a network without reservations is used. By automatically requesting network reservations at run-time we have enabled any application to transparently and painlessly use dedicated high speed reservable networks to increase communication performance. VRESERVE operates as follows, after VNET has decided which overlay links to create, but before it has created them, VRESERVE analyzes each link to determine if it can be better served using a reservation. A key point is that we create an overlay link on top of the reserved path. At first glance this may seem to be redundant, but it allows us to use VNET to perform routing. Without the overlay we would be forced to modify the host machines' routing tables or rewrite the packet headers. With the overlay in place, however, we can perform routing transparently. More details can be found elsewhere [59]. VRESERVE is joint work with my fellow graduate student, John R. Lange and our advisor Peter A. Dinda.

- **CPU reservation:** My fellow graduate student, Bin Lin, has designed and implemented a system, VSched, that allows reservation of CPU in terms of period and slice [60]. VSched uses the periodic real-time scheduling model to reserve CPU period and slice for VMs executing in the system. This allows for an additional adaptation mechanism. Currently, Vsched is not integrated with VNET. I intend to take CPU reservations into account while modeling the complete abstract problem.

6.2 Measure of application's performance

Beyond application demand inference, network measurements and adaptation mechanisms, we need to have some measure of the application's performance, in my thesis work, I will attempt to look at applications that have different measures of application performance. Maximizing throughput might be important for some applications while reducing latency might be crucial for others.

6.3 Adaptation metrics

Depending on the problem formulation a number of objective functions can be defined. It is a claim that optimizing (maximizing / minimizing) the appropriate objective function will result in improved application performance. In this thesis, I will come up with such an objective function and study its effectiveness for a wide range of distributed applications.

6.4 Adaptation algorithms

The adaptation control algorithms are implemented in the VADAPT component of Virtuoso. The full control problem, informally stated in English, is "Given the network traffic load matrix of the application

and its computational intensity in each VM, the topology of the network and the load on its links, routers, and hosts, what is the mapping of VMs to hosts, the overlay topology connecting the hosts, and the forwarding rules on that topology, the required CPU and network reservations, that maximizes the application performance?" Once the problem has been formalized and an appropriate optimization metric defined, I will characterize the computational complexity of the problem. I expect the problem to be NP-complete. I will attempt to characterize it more specifically with an eye to obtain approximate algorithms as solutions. The initial ground work for the formalization of the adaptation control problem is discussed in Section 7.

7. Problem formulation

In this section we lay the ground work for formalizing the complete general adaptation problem in virtual execution environments. The formalization is a first step towards understanding the problem better. The adaptation problem has three main components, measured data (application demands, network measurements, etc.), adaptation mechanisms leveraged (topology changes, routing, VM migration, etc.) and the objective function to be optimized. At this point in time, I am not sure if I will include the hosts' compute and capacity constraints, the VMs' compute and capacity demands, application's latency demands and the network's latency measurements in the problem formulation. This is because, currently, my system does not measure this information. For the purpose of this document, I am working with the assumption that I will measure that information as well. However, I might drop some of them from the formulation in the course of my thesis work.

I have designed a framework for the formalization, however several pieces are currently black boxes which I will design as part of my thesis work. Figure 8 is an illustration of this framework, the darkened boxes are the pieces that are yet to be designed.

7.1 Problem description

VNET monitors the underlying network and provides a directed VNET topology graph, $G = (H, E)$, where H are VNET nodes (hosts running VNET daemons and capable of supporting one or more VMs) and E are the possible VNET links. Note that this may not be a complete graph as many links may not be possible due to particular network management and security policies at different network sites. VNET also provides estimates for the available bandwidth and latencies over each link in the VNET topology graph. These estimates are described by a bandwidth capacity function, $\mathrm{bw} : E \rightarrow \mathbb{R}$, and a latency function, $\mathrm{lat} : E \rightarrow \mathbb{R}$.

In addition, VNET also collects information regarding the space capacity (in bytes) and compute capacity made available by each host, described by a host compute capacity function, $\mathrm{compute} : H \rightarrow \mathbb{R}$ and a host space capacity function, $\mathrm{size} : H \rightarrow \mathbb{R}$. The set of virtual machines participating in the application is denoted by the set VM. The size and compute capacity demand made by every VM is also estimated, denoted by a VM compute demand function, $\mathrm{vm\_compute} : \mathrm{VM} \rightarrow \mathbb{R}$ and a VM space demand function, $\mathrm{vm\_size} : \mathrm{VM} \rightarrow \mathbb{R}$, respectively.

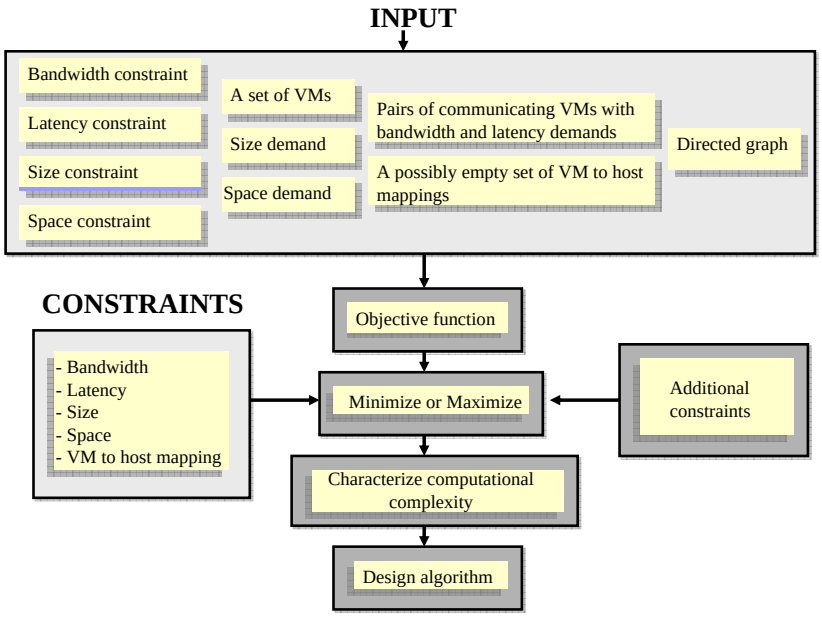

Figure 8. Framework for problem formulation of the adaptation problem.

VTTIF infers the application communication topology to generate the traffic requirements of the application, $\mathcal{A}$, which is a set of 4-tuples, $A_i = (s_i, d_i, b_i, l_i)$, $i = 1, 2 \ldots m$, where s_i is the source VM, d_i is the destination VM, b_i is the bandwidth demand between the source destination pair and l_i is the latency demand between the source destination pair.

The goal is to find an adaptation algorithm that uses the measured and inferred data and drives the adaptation mechanisms at hand to improve application performance. In other words we wish to find

- a mapping from VMs to hosts, $\mathrm{vmap} : \mathrm{VM} \rightarrow H$, meeting the size and compute capacity demands of the VMs within the host constraints. Further, we may also be given a set of additional constraints such as reservation requirements and mappings from VMs to hosts that have to be maintained at all times, represented by a set of ordered pairs $M_i = (\mathrm{vm}_i, h_i)$, $\mathrm{vm}_i \in \mathrm{VM}$, $h_i \in H$.

- a routing, $R : \mathcal{A} \rightarrow \mathcal{P}$, where $\mathcal{P}$ is the set of all paths in the graph $G = (H, E)$,i.e. for every 4-tuple, $A_i = (s_i, d_i, b_i, l_i)$, allocate a path, $p\big(\mathrm{vmap}(s_i), \mathrm{vmap}(d_i)\big)$, over the overlay graph, G, meeting the application demands while satisfying the bandwidth and latency constraints of the network.

Next, we will need to define an objective function, with the claim that optimizing the same (maximizing or minimizing) results in improved application performance while meeting some additional constraints. At this point in time the objective function and the additional constraints are black boxes.

7.1.1 Objective functions:

There are a number of objective functions which could be defined. The key is to find the one which will work effectively for a wide range of distributed applications. Following are some that I am thinking about.

Single-hop routing: In this we adapt the application to the network measured, using the adaptation mechanisms described above, such that each pair of communicating VMs are separated by at most a single overlay hop. Note that multiple VMs can be mapped onto a single VNET daemon.

Residual bottleneck bandwidth: We infer the application demands using VTTIF, measure the underlying network using Wren and attempt to find a mapping of VMs to Hosts and communicating pairs of VMs to paths over the VNET network such that after all the mappings the sum of the residual bottleneck bandwidths over all the mapped paths is maximized. The residual bandwidth over the VNET network is defined as the difference between its capacity and the sum of the demands mapped onto it. The bottleneck bandwidth of a path over the VNET graph is defined as the minimum residual bandwidth over all the VNET edges participating in that path. The intuition behind this metric is to leave the most room for the application to increase performance within the current configuration.

Additional metrics: There are a number of additional metrics that could be defined, such as minimizing the residual bottleneck capacity, this would create a kind of "tight fit" thereby increasing room for other applications to enter the system. Though, in previous work, I have proposed and evaluated the two metric in the context of two classes of distributed applications and found both the metrics to be effective, it is not obvious if there exists a single metric and a single optimization scheme that is effective for a wide range of distributed applications.

User guided metric: There is a second dimension to thinking about objective functions. The objective function could be defined or refined, as the case may be, using direct user input. My fellow graduate student, Bin Lin, has built a system, VSched, that allows the user to define the objective function [60]. However, my optimization scheme will not cover such objective functions.

7.1.2 Additional constraints

I will refine the problem formulation by taking into consideration additional constraints. Taking everything into account while designing the optimization metric can lead to a very complex formulation that may not yield itself to a good approximate solution. Hence consideration of these additional constraints will lead to a more practical problem formulation.

7.2 Algorithms

As part of my thesis work, I will complete the formalization framework developed for the adaptation problem. Once I have decided on an objective function and the additional constraints, I will analyze the formulated problem and characterize its computational complexity. I expect the problem to be NP-complete, if that is indeed the case I will design an approximate algorithm as a solution.

8. A possible problem formulation using residual bottleneck bandwidth as a metric

To get a sense of the final problem formulation and its computational complexity, I plugged in the residual bottleneck bandwidth based objective function in the formulation framework to obtain a possible problem formulation [88]. This was joint work done with my fellow graduate students, Manan Sanghi, John R. Lange and my advisor Peter A. Dinda. It should be noted that the problem formulation in my thesis work might have a different objective function along with some additional constraints. I have looked at a specific, possible case just to get a sense of the theoretical issues involved in the adaptation problem. Here, I do not take into account any additional constraints.

8.1 Residual bottleneck bandwidth

Once all the mappings and paths have been decided, each VNET edge will have a residual capacity, rc_e, which is the bandwidth remaining un-utilized on that edge, in that direction.

$$\mathrm{rc}_e = \mathrm{bw}_e - \sum_{e \in R(A_i)} b_i$$

.

For each mapped path, $R(A_i)$, we define its bottleneck bandwidth, $\mathrm{bb}\big(R(A_i)\big) = \min_{e \in R(A_i)} \big\{\mathrm{rc}_e\big\}$ and its total latency, $\mathrm{tl}\big(R(A_i)\big) = \sum_{e \in R(A_i)} \big(lat_e\big)$

The aim in this specific case is to maximize the sum of residual bottleneck bandwidths over each mapped path.

8.2 Problem formulation

Problem 1 (Generic Adaptation Problem In Virtual Execution Environments (GAPVEE))

INPUT:

- *A directed graph* $G = (H, E)$
- *A function* bw *:* $E \rightarrow \mathbb{R}$
- *A function* lat *:* $E \rightarrow \mathbb{R}$
- *A function* $\mathrm{compute}$ *:* $H \rightarrow \mathbb{R}$
- *A function* size *:* $H \rightarrow \mathbb{R}$

- *A set,* $\mathrm{VM} = (\mathrm{vm}_1, \mathrm{vm}_2 \ldots \mathrm{vm}_n)$, $n \in \mathbb{N}$
- *A function* $\mathrm{vm\_compute}$ *:* $\mathrm{VM} \rightarrow \mathbb{R}$
- *A function* $\mathrm{vm\_size}$ *:* $\mathrm{VM} \rightarrow \mathbb{R}$
- *A set of ordered 4-tuples* $\mathcal{A} = \{(s_i, d_i, b_i, l_i) \mid s_i, d_i \in \mathrm{VM};\ b_i, l_i \in \mathbb{R};\ i = 1, \ldots, m\}$
- *A set of ordered pairs* $\mathcal{M} = \{(\mathrm{vm}_i, h_i) \mid \mathrm{vm}_i \in \mathrm{VM}, h_i \in H;\ i = 1, 2 \ldots r, r \leq n\}$

OUTPUT: $\mathrm{vmap} : VM \rightarrow H$ *and* $R : \mathcal{A} \rightarrow \mathcal{P}$ *such that*

- $\sum_{\mathrm{vmap(vm)}=h} \big(\mathrm{vm\_compute(vm)}\big) \leq \mathrm{compute(h)}, \forall\, h \in H$
- $\sum_{\mathrm{vmap(vm)}=h} \big(\mathrm{vm\_size(vm)}\big) \leq \mathrm{size}(h), \forall\, h \in H$
- $h_i = \mathrm{vmap}(\mathrm{vm}_i), \forall M_i = (\mathrm{vm}_i, h_i) \in \mathcal{M}$
- $\big(bw_e - \sum_{e \in R(A_i)} b_i\big) \geq 0, \forall e \in E$
- $\big(\sum_{e \in R(A_i)} \mathrm{lat}_e\big) \leq l_i, \forall e \in E$
- $\sum_{i=1}^{m} \big(\min_{e \in R(A_i)} \{\mathrm{rc}_e\}\big)$, *where* $\mathrm{rc}_e = (\mathrm{bw}_e - \sum_{e \in R(A_i)} b_i)$, *is maximized*

8.3 A special case of the specific adaptation problem

The generic adaptation problem formulated seeks a mapping, vmap from VMs to hosts and routing, R of VM traffic over the overlay network, G. To establish the hardness of the problem, we consider a special case of the problem wherein all the VM to host mappings are constrained by the ordered pairs $\mathcal{M}$ and latency demands are dropped, leaving us only with the routing problem.

Since the mappings are pre-defined, we can formulate the problem in terms of only the hosts and exclude all VMs. Also, as the latency demands have been dropped, the application 4-tuple reduces to 3-tuple, $A_i = (s_i, d_i, b_i)$, $s_i, d_i \in H$, $b_i \in \mathbb{R}$, $i = 1, 2 \ldots m$. Notice that now $s_i, d_i \in H$ as VM to host mappings are fixed and VMs are synonymous with the hosts that they are mapped to.

Problem 2 (Routing Problem In Virtual Execution Environments (RPVEE))

INPUT:

- *A directed graph* $G = (H, E)$
- *A function* bw *:* $E \rightarrow \mathbb{R}$
- *A set of ordered 3-tuples* $\mathcal{A} = \{(s_i, d_i, b_i) \mid s_i, d_i \in H;\ b_i \in \mathbb{R};\ i = 1, \ldots, m\}$

OUTPUT: $R : \mathcal{A} \rightarrow \mathcal{P}$ *such that*

- $\big(\mathrm{bw}_e\big) - \big(\sum_{e \in R(A_i)} b_i\big) \geq 0, \forall e \in E,$
- $\sum_{i=1}^{m} \big(\min_{e \in R(A_i)} \{\mathrm{rc}_e\}\big)$, *where* $\mathrm{rc}_e = (\mathrm{bw}_e - \sum_{e \in R(A_i)} b_i)$, *is maximized*

8.4 Computational complexity

Theorem 1 *RPVEE is NP-hard.*

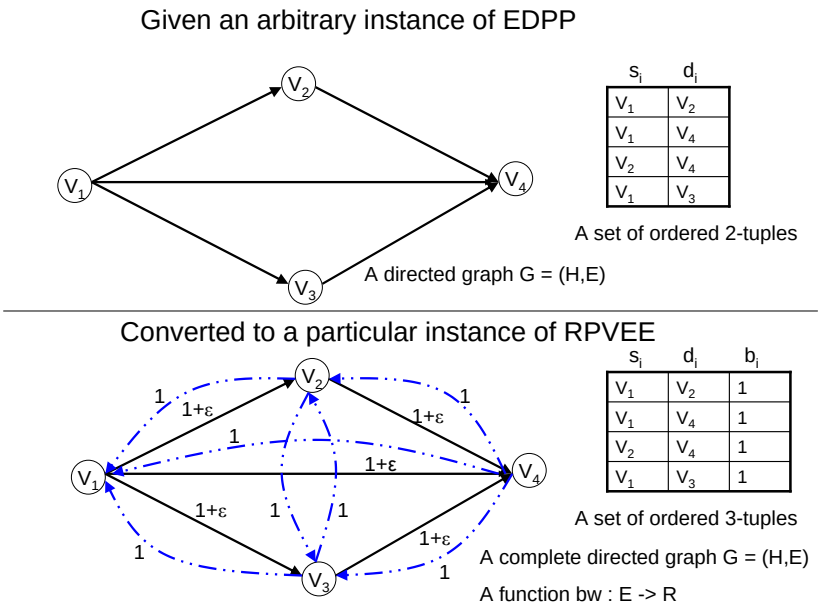

Figure 9. Reducing EDPP to RPVEE. The edge weights are bandwidths as specified by the function bw.

The NP-hardness for the problem is established by reduction from the Edge Disjoint Path Problem (EDPP) which is shown to be NP-complete [52]. In the interest of space we provide only a brief sketch of the reduction here. The complete proof for Theorem 1 can be found in Appendix A.

The edge disjoint path problem (EDPP) is specified as:

Problem 3 (Edge Disjoint Path Problem (EDPP))

INPUT:

- *A graph* $G = (H, E)$, $|H| = p$, $|E| = q$
- *A set of ordered 2-tuples* $\mathcal{S} = \{(s_i, d_i) \mid s_i, d_i \in H;\ i = 1, \ldots, k\}$

OUTPUT:

- *Yes, if and only if* $\forall (s_i, d_i) \in \mathcal{S}$ *their exist edge disjoint paths from* s_i *to* d_i *in* $G = (H, E)$
- *No, otherwise*

For reducing EDPP to an instance of the decision version of RPVEE, construct a complete graph $G' = (V, E')$ where $\text{bw}((u, v)) = 1+\epsilon$ if $(u, v) \in E$ and $\text{bw}((u, v)) = 1$ if $(u, v) \notin E$. Further for all $(s_i, d_i) \in \mathcal{S}$, let $(s_i, d_i, 1) \in \mathcal{A}$. Figure 9 illustrates this reduction. Note that there exists edge disjoint paths for the EDPP if and only if the sum of bottleneck bandwidths in the instance of RPVEE is $k{\cdot}\epsilon$.

Since GAPVEE is a special case of RPVEE, the following theorem immediately follows.

Theorem 2 *GAPVEE is NP-hard.*

The complete proofs for Theorem 2 can be found in Appendix A.

9. Preliminary results

I carried out a series of experiments to understand the overheads of my system and to study the effectiveness of the adaptation mechanisms in the context of automatic, dynamic and run-time adaptation. I present some of these results in this section. Some of this work is joint with my fellow graduate students Ashish Gupta, John R. Lange and our advisor Peter A. Dinda.

9.1 Applications

For this preliminary study we used two classes of distributed applications:

- **Patterns: A bulk synchronous parallel (BSP) benchmark:** It is a synthetic workload generator that captures the computation and communication behavior of BSP programs. Patterns emulates a BSP program with alternating dummy compute phases and communication phases according to the chosen topology, operation, and compute/communicate ratio. In particular, we can vary the number of nodes, the compute/communicate ratio of the application, and select from communication operations such as reduction, neighbor exchange, and all-to-all on application topologies.
- **TPC-W: A transactional web ecommerce benchmark:** It is an industry benchmark for transactional web ecommerce applications. TPC-W models an online bookstore [80]. This benchmark was used to study the effectiveness of our approach for a non-parallel application.

9.2 System overheads

It is very important to study and understand the overheads of such a middleware system. The overheads must be negligible when compared to the benefits obtained by automatic, dynamic and run-time adaptation.

9.2.1 Configuration time of the adaptive system

Figure 10 shows the time required to create different VNET topologies among eight VNET daemons each hosting a single VM. Here, all the hosts are in single cluster (IBM e1350, nodes are dual 2.0 GHz Xeons with 1.5 GB RAM running Red Hat Linux 9.0 and VMware GSX Server 2.5, connected by a 100 mbit switched network). The Proxy and the user are located on a network separated by a metropolitan area network (MAN). We will refer to this setup as the single cluster setup. This setup helps emphasize overheads and eliminate other factors such as wide area latency, etc.

It takes 0.94 seconds to create the resilient star topology among the VNET daemons, including time to add the links and populate the forwarding tables. It takes a further 1.6 seconds to add all the fast path links and corresponding forwarding rules for an all-to-all topology. Adding fast path links for a bus topology takes

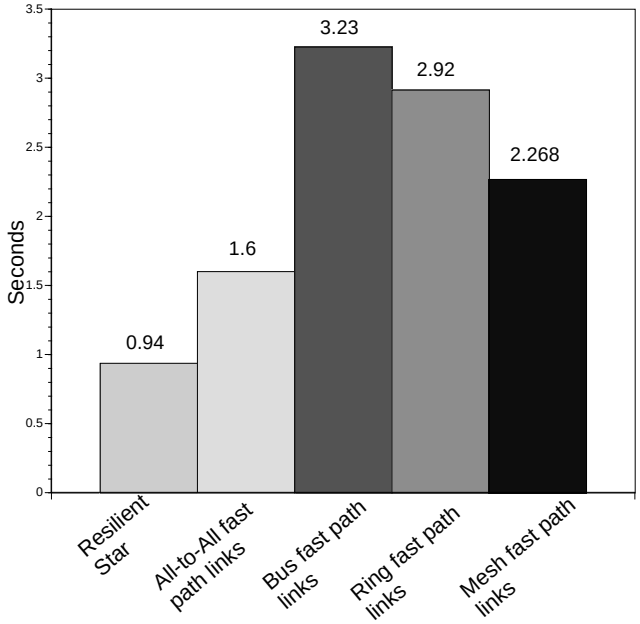

Figure 10. Time to set up the backbone star configuration and to add fast path links for different (inferred) topologies.

longer (3.23 seconds), even though there are fewer links. This is because VNET does not use hierarchical routing. Since VNET operates at the link layer, virtual machine migration would punch holes in hierarchical routing tables. Hence, VNET forwards packets based on a source and destination address match rather than just the destination address match, which leads to an increase in the number of forwarding rules for some topologies such as the bus topology.

9.2.2 Overheads of the reservation system

To evaluate our reservation system, we use the OMNInet network [47] and the ODIN [65] light path reservation system. Figure 11 shows the physical topology of a section of OMNInet. OMNInet is an experimental fully connected network that spans several sites in Chicago. We use machines directly connected to the optical switches at two of the sites. OMNInet is run by the International Center for Advanced Internet Research (iCAIR). More details can be found elsewhere [59]. This is joint work with my fellow graduate student, John R. Lange and our advisor Peter A. Dinda.

Figure 12 shows the costs involved in configuring the network using our system. The primary cost was the time spent in the reservation system itself. Creating a path entails two delays. The first is a software delay. It took $\sim$2.5 seconds for ODIN to send the configuration commands to all the switches. The second delay ($\sim$15 seconds) is the time needed for the hardware to reconfigure itself and for the path to stabilize. This stabilization delay is constant, regardless of the complexity of the number of switches being configured. The software delay, however, will grow linearly with the number of switches in a path because ODIN does not

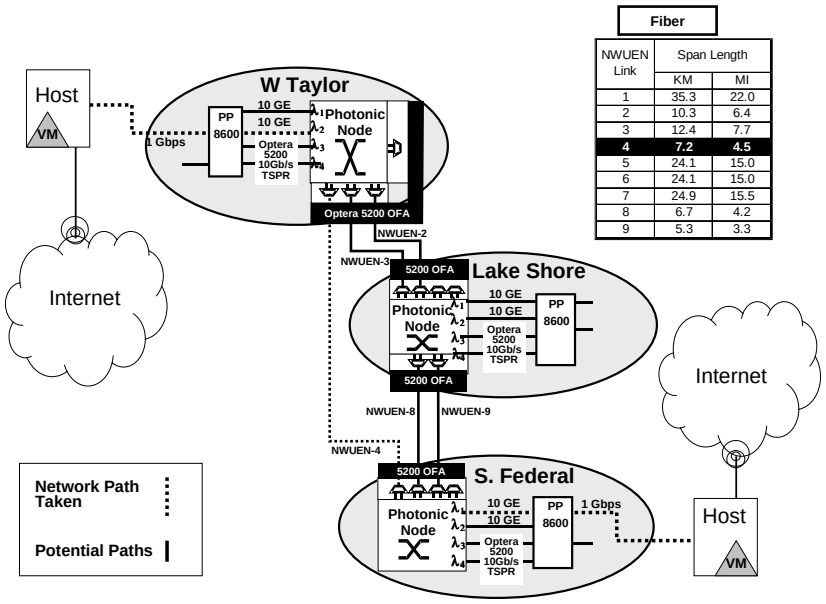

Figure 11. Physical topology of the OMNInet testbed network.

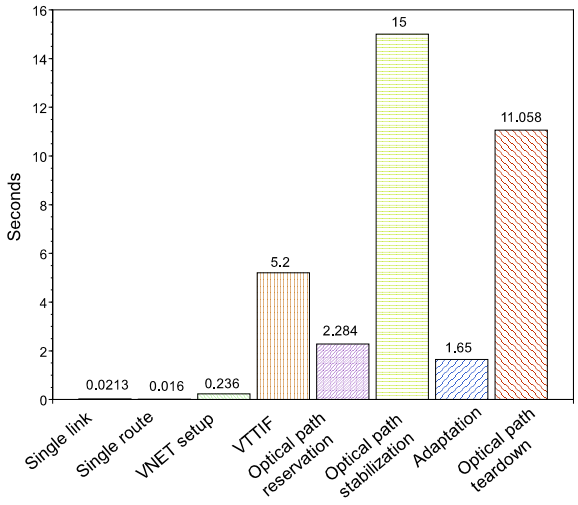

Figure 12. The configuration-time costs for the two VM scenario shown in Figure 11 executing Patterns with all-to-all communication.

currently support parallel configuration. The time taken to tear down an optical path was $\sim$12 seconds.

VTTIF is a significant, but smaller contributor to the setup delay. It must observe traffic for some period of time before it can report a topology. The VTTIF time is a function of a number of parameters and can be as low as one second. The time to execute VRESERVE and to create the individual overlay links is lost in the noise.

The total time from the start of network communication to channeling packets over an overlay link running through a lightpath is < 30 seconds.

9.2.3 Overheads of the migration system

Virtuoso allows us to migrate a VM from one physical host to another. Though our migration scheme is not the fastest, much work exists that demonstrates that fast migration of VMs running commodity applications and operating systems is possible [70, 73, 57]. Migration times down to 5 seconds have been reported [57]. As migration times decrease, the rate of adaptation we can support and my work's relevance increases.

9.3 Effectiveness of adaptation mechanisms

The framework for the complete adaptation problem has been described in Section 7. In the course of my thesis work I will complete the problem formulation, characterize its complexity and design an algorithm. From my experience in designing the framework, I have the sense that only approximate solutions will be possible.

Here, I present some initial results that suggest that the adaptation mechanisms are highly effective even when driven by very simple greedy heuristics. The problem formulation is a special case of the more generic incarnation described previously. Specifically, latency, space and size constraints are not considered. The objective function is to map communicating VMs on hosts connected by high bandwidth links and to then match the overlay topology to the communication topology as inferred by VTTIF. This is a simple heuristic approach, yet the results are very promising. The comparison here is between adaptation in virtual environments versus no adaptation in virtual environments.

9.3.1 Informal problem description

The inputs to the problem are

- A graph representing the application topology of the VMs and a traffic load matrix among them
- A matrix representing the available bandwidth among the Hosts running VNET daemons

VADAPT's goal is to use this information to choose a configuration that maximizes the performance of the application running inside the VMs. A configuration consists of

- The mapping of VMs to Hosts running VNET daemons

- The topology of the VNET overlay network
- The forwarding rules on that topology

The heuristic is that the application's performance improves as compared to the case with no adaptation if we do the following greedily:

- Map communicating VMs onto hosts that connected by high bandwidth links
- Match the overlay topology to the communication topology as inferred by VTTIF

9.3.2 Greedy heuristic

The greedy heuristic that I evaluated consisted of two sequential steps, map virtual machines onto physical hosts and next to adapt the VNET overlay topology to the communication behavior of the application.

The traffic intensity matrix inferred by VTTIF is represented as an adjacency list where each entry describes communication between two VMs. Further, the throughput estimates between each pair of VNET daemons arranged in decreasing order are used. The heuristic is as follows:

1. Generate a new list which represents the traffic intensity between VNET daemons that is implied by the VTTIF list and the current mapping of VMs to hosts.
2. Order the VM adjacency list by decreasing traffic intensity.
3. Order the VNET daemon adjacency list by decreasing throughput.
4. Make a first pass over the VM adjacency list to locate every non-overlapping pair of communicating VMs and map them greedily to the first pair of VNET daemons in the VNET daemon adjacency list which currently have no VMs mapped to them. At the end of the first pass, there is no pair of VMs on the list for which neither VM has been mapped.
5. Make a second pass over the VM adjacency list, locating, in order, all VMs that have not been mapped onto a physical host. These are the "stragglers".
6. For each of these straggler VMs, in VM adjacency list order, map the VM to a VNET daemon such that the throughput estimate between the VM and its already mapped counterpart is maximum.
7. Compute the differences between the current mapping and the new mapping and issue migration instructions to achieve the new mapping.

The topology adaptation heuristic is as follows:

1. Generate a new list which represents the traffic intensity between VNET daemons that is implied by the VTTIF list and the current mapping of VMs to hosts.

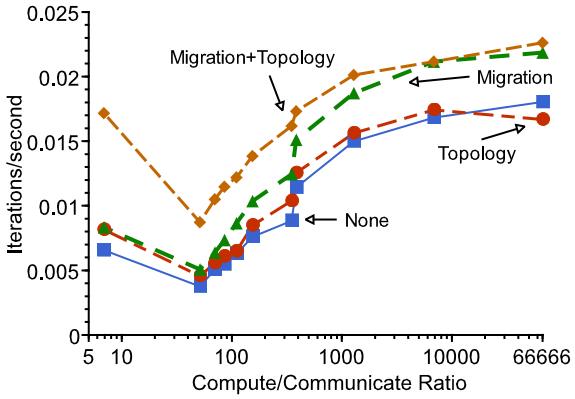

Figure 13. Effect on application throughput of adapting to compute/communicate ratio.

2. Order this list by decreasing traffic intensity.

3. Establish the links in order.

The precise details of these algorithms can be found in Appendix B.

9.3.3 Results for a BSP benchmark

We studied the following scenarios:

- Adapting to compute/communicate ratio: Patterns was run in 8 VMs spread over the WAN (4 on Northwestern's e1350, 3 on the slower Northwestern cluster, and 1 at CMU). The compute/communicate ratio of patterns was varied.

- Adapting to external load imbalance: Patterns was run in 8 VMs all on Northwestern's e1350. A high level of external load was introduced on one of the nodes of the cluster. The compute/communicate ratio of patterns was varied.

In both cases, patterns executed an all-to-all communication pattern. For an application with a low compute/communicate ratio, we would expect that migrating its VMs to a more closely coupled environment would improve performance. We would also expect that it would benefit more from topology adaptation than an application with a high ratio.

Figure 13 illustrates our scenario of adapting to the compute/communicate ratio of the application. For a low compute/communicate ratio, we see that the application benefits the most from migration to a local cluster and the formation of the fast path links. In the WAN environment, adding the overlay links alone doesn't help much because the underlying network is slow. Adding the overlay links in the local environment has a dramatic effect because the underlying network is much faster.

As we move towards high compute/communicate ratios migration to a local environment results in significant performance improvements. The hosts that we use initially have diverse performance characteristics.

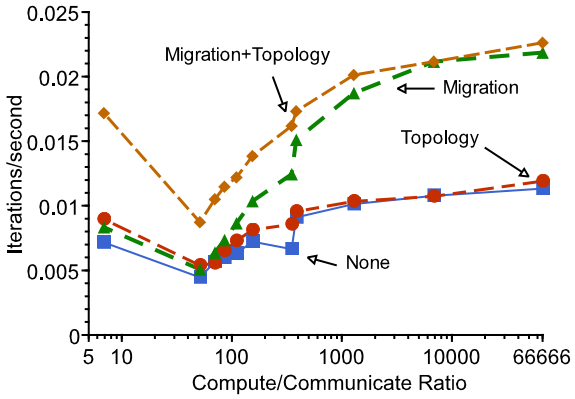

Figure 14. Effect on application throughput of adapting to external load imbalance.

This heterogeneity leads to increasing throughput differences as the application becomes more compute intensive. Because BSP applications run at the speed of the slowest node, the benefit of migrating to similar-performing nodes increases as the compute/communicate ratio grows.

Figure 14, shows the results of adapting to external load imbalance. We can see that for low compute/communicate ratios, migration alone does not help much. The VMs are I/O bound here and do not benefit from being relieved of external CPU load. However, migrating to a lightly loaded host *and* adding the fast path links dramatically increases throughput. After the migration, the VM has the CPU cycles needed to drive network much faster.

As the compute/communicate ratio increases, we see that the effect of migration quickly overpowers the effect of adding the overlay links, as we might expect. Migrating the VM to a lightly loaded machine greatly improves the performance of the whole application.

9.3.4 Results for a transactional web ecommerce benchmark

The purpose of this study was to evaluate the effectiveness of our approach for non-parallel applications. Most web sites serve dynamic content and are built using a multi-tier model, including the client, the web server front end, the application server(s), cache(s), and the database. It should be noted that these results are very preliminary. This is joint work with my fellow graduate student, Ashish Gupta and our advisor Peter A. Dinda.

TPC-W models an online bookstore. The separable components of the site can be hosted in separate VMs. Figure 15 shows the configuration of TPC-W that we use, spread over four VMs hosted on our e1350 cluster. Remote Browser Emulators (RBEs) simulate users interacting with the web site. RBEs talk to a web server (Apache) that also runs an application server (Tomcat). The web server fetches images from an NFS-mounted image server, alternatively forwarding image requests directly to an Apache server also running on the image server. The application server uses a back-end database (MySQL) as it generates content. We run the browsing interaction job mix (5% of accesses are order-related) to place pressure on the front-end web

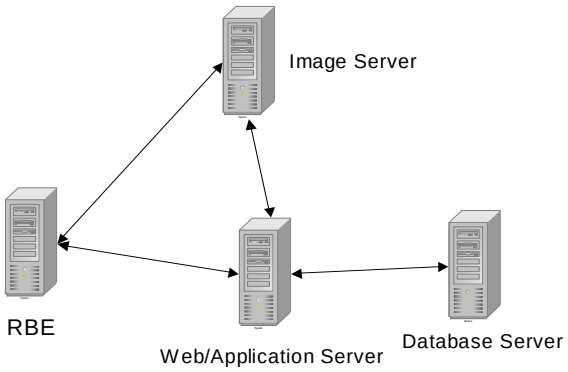

Figure 15. The configuration of TPC-W used in our experiment.

	No Topology	Topology
No Migration	1.216	1.76
Migration	1.4	2.52

Figure 16. Web throughput (WIPS) with image server facing external load under different adaptation approaches.

servers and the image server.

The primary TPC-W metric is the WIPS rating. Figure 16 shows the sustained WIPS achieved under different adaptation approaches. We are adapting to a considerable external load being applied to the host on which the image server is running. When VADAPT migrates this VM to another host in the cluster, performance improves. Reconfiguring the topology also improves performance as there is considerable traffic outbound from the image server. Using both adaptation mechanisms simultaneously increases performance by a factor of two compared to the original configuration.

9.4 Scaling

We tested topology adaptation scenarios with all-to-all traffic among up to 28 VMs, the maximum possible on a single one of our clusters. We used a pre-defined VM to host mapping to study the scalability of the overlay adaptation. While the cost of VM migration to meet an adaptation goal grows with the number of VMs, the number of links in the overlay topology can grow with the square of the number VMs, thus the system will scale as VNET scales, not as migration scales. The number of forwarding rules per node can also grow with the square of the number of VMs, although the worst topology for this is a linear one, which is unlikely to be used. For an all-to-all, it grows linearly with the number of VMs.

At 28 VMs, we can create our initial star topology in about about 2.9 seconds, with 84% of the time spent loading forwarding rules into VNET daemons. The total number of links and forwarding rules in the system for a star grows linearly with the number of VMs. Adding the full all-to-all topology takes 20.5 seconds, of which 67% involves loading forwarding rules. The inference time remains roughly the same as with the

Activity	Artifacts	Duration
Porting applications	distributed applications	3 months
Problem formulation	optimization metric	1 month
Algorithm development	adaptation algorithm	4 months
Evaluation	implementation, results	2 months
Writing/defense	defended thesis	2 months
Total time	dissertation	12 months

Figure 17. Thesis timeline.

smaller scenarios we described previously.

Not surprisingly, the benefit of adapting the topology to the application grows as the number of VMs grows.

10. Outline of thesis

In my thesis I will generalize my work on adaptation in virtual execution environments to a wide range of distributed applications and come up with a single optimization algorithm that would benefit them all. There will be five stages to this work. In the first stage I will explore diverse distributed applications and port them to my virtual execution environment. At this point in time we have already ported four distributed application benchmarks, I am looking to accumulate around 10 benchmarks and real applications. In the second stage of this work, I will analyze the performance of these applications, study the adaptation problem and come up with a single metric to be optimized. The bulk of my time will be spent in formalizing the adaptation problem using the metric that I come up and analyzing its computational complexity. If it turns out to be NP-hard as I expect, I will design approximate algorithms to drive the adaptation mechanisms. This will be the third stage of my work. In the fourth stage, I will implement the adaptation algorithm in my system and evaluate all the distributed applications to determine the effectiveness of the adaptation algorithm. In the final stage, I will write the dissertation document. The remainder of this section elaborates on the various stages. A timeline for the thesis work is given in Figure 17.

10.1 Porting applications

Virtuoso provides a wide-area distributed computing environment that can, in theory, scale to sizes even bigger than the current Internet. There are a wide range of such distributed applications, most of them can be classified into four main categories [6]

- **Inherent distributed nature:** applications that exploit their inherent distributed nature
- **High turnaround time:** huge applications whose turnaround time can be decreased by execution in such environments

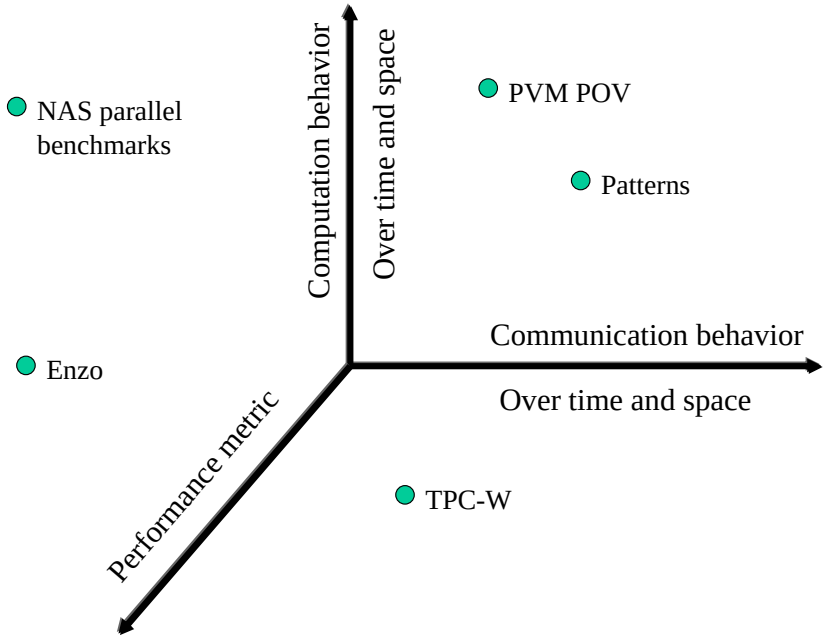

Figure 18. Characterization of distributed applications.

- **Large applications:** applications whose execution is outside the capabilities of a single (sequential or parallel) architecture
- **Exploit distributed resources:** applications that require certain resources that are geographically distributed under different administrative domains

Distributed applications typically differ in their compute and communication behaviors over space and time. Further, the measure of the application's performance, such as bandwidth or latency differs as well. Figure 18 shows this characterization.

A key point in my thesis is that adaptation mechanisms made available by virtual machines interconnected with a virtual network can be effective for a wide range of distributed applications with different compute and communication behavior over time and space. I will port a diverse set of applications and benchmarks, a few in each of the above categories, to my execution environment to study the effectiveness of application independent adaptation.

I already have the following application benchmarks running in my execution environment:

- **Patterns:** It is a synthetic workload generator that captures the computation and communication behavior of BSP programs. In particular, we can vary the number of nodes, the compute/communicate ratio of the application, and select from communication operations such as reduction, neighbor exchange, and all-to-all on application topologies including bus, ring, n-dimensional mesh, n-dimensional torus, n-dimensional hypercube, and binary tree. Patterns emulates a BSP program with alternating

dummy compute phases and communication phases according to the chosen topology, operation, and compute/communicate ratio.

- **NAS parallel benchmarks (NPB):** These were devised by the Numerical Aerodynamic Simulation (NAS) program of the National Air and Science Administration (NASA) for the performance evaluation of highly parallel computers. While the NPB is rooted in the problem of computational fluid dynamics and computational aerosciences, they are valuable for our evaluation purposes as they are rigorous and very close to real applications and are very widely used. I will use the PVM implementations of the NAS benchmarks [5], IS, MG, FT, and EP as developed by White et al. [97]. This application was ported by my fellow graduate student, Ashish Gupta.

- **PVM POV:** The Persistence of Vision Ray-Tracer creates three-dimensional, photo-realistic images using a rendering technique called ray-tracing. It reads in a text file containing information describing the objects and lighting in a scene and generates an image of that scene from the view point of a camera also described in the text file. Ray-tracing is not a fast process by any means, but it produces very high quality images with realistic reflections, shading, perspective and other effects. It is a time intensive task and will be very useful in studying the effectiveness of our adaptation scheme. I will use the PVM version of the popular ray tracer POVRAY. The PVM version gives it the ability to distribute a rendering across multiple heterogeneous systems [21]. This application was ported by my fellow graduate student, Ashish Gupta.

- **TPC-W:** It is an industry benchmark[5] for such sites. TPC-W models an online bookstore. The separable components of the site can be hosted in separate VMs. Figure 15 shows one possible configuration of TPC-W. Remote Browser Emulators (RBEs) simulate users interacting with the web site. RBEs talk to a web server (Apache) that also runs an application server (Tomcat). The web server fetches images from an NFS-mounted image server, alternatively forwarding image requests directly to an Apache server also running on the image server. The application server uses a back-end database (MySQL) as it generates content. This application adds variety as its functionality and requirements are very different than typical scientific applications and will contribute to the breadth of my study. This application was ported by my fellow graduate student, Ashish Gupta.

I will spend a considerable amount of time in finding other real (as opposed to benchmarks) distributed applications for the purposes of this thesis. One application that I have in mind is

Enzo: It is an adaptive mesh refinement (AMR), grid-based hybrid code (hydro + N-Body) which is designed to do simulations of cosmological structure formation. It uses the algorithms of Berger and Colella [9] to improve spatial and temporal resolution in regions of large gradients, such as gravitationally collapsing objects. The Enzo simulation software is incredibly flexible, and can be used to simulate a wide range of cosmological situations. Enzo has been parallelized using the MPI message-passing library and can run on

[5]We use the Wisconsin PHARM group's implementation [42], particularly the distribution created by Jan Kiefer.

any shared or distributed memory parallel supercomputer or PC cluster making it ideal for evaluating our wide-area distributed computing system. Enzo is publicly available [13].

10.2 Formalizing the adaptation problem

In this stage I will propose a single optimization metric, formulate the adaptation problem in the context of the proposed metric and characterize its computational complexity.

10.2.1 Proposing a single optimization metric

I will execute all the ported applications in my execution environment with the aim of coming up with a single optimization metric for all of them. Based on my experience with the applications that I have already ported and the adaptation problem at hand, I have a couple of metrics in mind. Two of them are:

- I will map VMs to hosts and communicating VMs to paths over the virtual network. Once the mapping has been determined we can define the residual bandwidth over each edge in the graph. The aim is to maximize the sum of residual bottleneck bandwidths over all the mapped paths. The intuition behind this metric is to leave the most room for the application to increase performance within the current configuration.
- To map VMs to hosts and communicating VMs to paths over the virtual network graph so as to minimize the sum of the residual bottleneck bandwidths over all the mapped paths. The intuition behind this is to produce a tightest fit so as to increase room for other applications to enter the system. This metric is more suitable for a system supporting multiple overlapping wide area applications.

At this point in time, I am not sure what the right metric is, I will carefully study the applications and analyze the typical conditions to come up with a single optimization metric. I have prior experience in such optimization problems. In my previous work, I have come up with optimization metrics and evaluated the same [85, 87, 38].

There are a variety of approaches that can be taken to adapt distributed applications. Most of these approaches differ in

- **Adaptation:** We can either adapt the application, the currently common approach, or we can do adaptation in a virtual execution environment requiring no modification to the application.
- **Control:** The adaptation control can be global or distributed.
- **Target:** The adaptation can be application specific or generic

My approach is to do adaptation in a virtual execution environment that requires no modification to the application. The adaptation control is global and the adaptation scheme is meant to be generic so as to be effective for a wide range of distributed applications. As any system scales, global control has to be

replaced with distributed control, hence adapting highly distributed large applications will necessarily require distributed control. In such a scenario, I expect a generic adaptation scheme to be comparable to application specific adaptation schemes.

Though, it is not a part of my thesis work, our group has plans to scale Virtuoso by distributing the adaptation control. Hence, I argue that in the long run designing a generic adaptation system is more effective than application specific schemes.

10.2.2 Problem formulation

I will then formalize the adaptation problem using the metric that I come up with and characterize its computational complexity. If it turns out to be NP-hard as I expect, I will design approximate heuristic algorithms to drive the adaptation mechanisms.

10.3 Algorithm development and simulation

In this stage I will take up the challenging task of developing an adaptation algorithm. At this point in time since the problem is not well defined, I do not have a good sense of the approach my work will take. I will consider exploring the combinatorial properties of the problem to arrive at an approximate solution. Another possible approach would be to form a linear programming problem and approximate it to its integer linear program version and then solving the same using solvers such as CPLEX [18].

After I have a working set of applications and a metric to optimize, I will revisit the adaptation problem and come up with a possibly modified problem formulation centered around the new optimization metric. I will then analyze the computational complexity of the problem. I suspect the problem will turn out to be NP-hard. I will then come up with an approximate algorithm and/or heuristics for the same.

I have prior experience in such theoretical analysis. Further, I already have a good sense of the related literature. I will leverage this previous experience in this algorithm development stage.

I will evaluate the quality of my heuristics using simulations. It is very important to get a sense of how close they are to the optimal but expensive solution. At this point in time, I am not quite sure how exactly I will go about it. One possibility would be to create small scenarios that lend themselves to exhaustive search in an attempt to come up with meaningful comparisons.

10.4 Evaluation

My evaluation will primarily focus on studying the effectiveness of adaptation mechanisms made available by virtual machines interconnected via virtual networks for a range of distributed applications. I will implement the algorithm developed in the previous stage in my execution environment. I will execute all the applications with and without adaptation mechanisms to study its effectiveness. I will also provide results as to the overheads of the system and study the acceptability of the overheads for wide area distributed computing. I will also classify the overheads in terms of implementation and design in an attempt to understand

the future work required to further improve performance and to characterize the limits of such a system, respectively.

10.5 Writing and defense

In the final stage of my thesis, I will document all my findings and defend them.

11. Thesis contributions

The contributions of my thesis will will include optimization models, algorithms, evaluations and artifacts. The contributions of my work up to this point include:

- **VNET:** I designed and implemented VNET, an Ethernet layer virtual network tool that creates and maintains the networking illusion, that the user's VMs are on the user's local area network (Section 3).
  - **Language for describing VNET topology:** I have designed a language for describing the VNET topology and forwarding rules. I defined a context free grammar for the language (Section 3.3).
  - **Parser for the VNET language:** I implemented a parser to parse descriptions in the VNET language. I have also implemented a variant of the parser that takes in high-level user requirements and generates a VNET topology description and associated forwarding rules in the VNET language (Section 3.3).
  - **VNET tools:** I designed and implemented a set of VNET tools that perform diverse functions such as setup, visualizing the current state of the system and reconfiguration (Section 3.3).
  - **VNET Evaluation:**

    I have carried out a detailed evaluation of VNET. To the best of my knowledge, this is the only existing detailed performance evaluation of such virtual networks.
    - **VNET performance:** I have evaluated VNET's performance with respect to throughput and latency. I carried out this evaluation for two separate cases, one with SSL encryption and one without. I compared its performance to the raw physical hardware and to a commercially available virtualization software. This was done for both LAN and WAN settings [86]. I adapted VNET for high speed networks by supporting creation of overlay links using UDP and by improving the forwarding rule lookup mechanism through a forwarding rule cache that gives us constant time lookup on average. All these enhancements have lead to a factor of three performance improvement, though there is still room for improvement (Section 3.4).
    - **VNET overheads:** I have also quantified the VNET overheads such as its setup and reaction times (Section 9.2.1).
    - **Scaling:** I have evaluated VNET to study how it scales with the increase in the number of VNET daemons and VMs serviced by them in terms of its overhead and the benefits of adaptation (Section 9.4). My important findings were:

- VNET scales in terms of the number of VNET links, forwarding rules and overheads with the number of VNET daemons and VMs supported by them.
- The cost of migration does not scale with the number of VMs in the system.
- The benefit of adapting the topology to the adaptation grows as the number of VMs grow.

- **Vision for adaptive environment:** We detailed the steps towards an adaptive virtual overlay network wherein un-modified applications running on top of un-modified operating systems could be adapted to available computational and network resources without the need for any user intervention [86].

- **Application independent adaptation mechanisms:** I have designed and implemented a second generation VNET, which includes support for arbitrary topologies and routing and provided adaptive control of the overlay (Section 3.2). To the best of my knowledge, currently no work exists for this.

- **Formalized generic adaptation problem:** I have designed a framework for formalization of the generic incarnation of the adaptation problem occurring in virtual execution environments (Section 7.1).

- **Defined a specific case of the generic adaptation problem:** I have a defined a specific, possible case of the generic adaptation problem where some of the constraints and requirements have been relaxed and a particular objective function used as an aid to gain better understanding of the problem (Section 8).

- **Analyzed its computational complexity:** I have carried out an analysis of the computational complexity of the above mentioned specific problem and found it to be NP-hard [88] (Section 8.4). This is joint work with Manan Sanghi, John R. Lange and Peter A. Dinda.

- **Adaptation metrics:** I have proposed and evaluated two metrics for the adaptation problem in adaptive virtual environments (Section 6.3 and Section 9). This is joint work with Ashish Gupta and Peter A. Dinda.

- **Adaptation heuristics:** I have proposed and evaluated greedy adaptation heuristics as solutions to specific versions of the adaptation problem (Section 9.3).

- **Adaptation Evaluation:** I studied benefits of the adaptation mechanisms made available by virtual machines connected via virtual networks. I found that for different applications the effectiveness of the adaptation mechanisms varied, but overall, the combined effect was significantly enhanced application performance (Section 9.3).

- **Evaluation using applications:**

  I have studied the effectiveness of the adaptation mechanisms in the context of the following two classes of application:

  - **BSP applications:** I found that for BSP applications the performance could be enhanced to up to a factor of two (Section 9.3.3).
  - **Multi-tier web sites:** I studied the benefits of adaptation in the context of non-parallel applications, specifically an industry benchmark for multi-tier web sites. This evaluation is still in its early stages, but the initial results are very promising and indicate that considerable performance gains are possible. This is joint work with Ashish Gupta and Peter A. Dinda (Section 9.3.4).

- **Automatic dynamic run-time optical network reservations:** To date very little work has been done on automatic network reservations based entirely on the applications needs at run time. Jointly with John R. Lange and Peter A. Dinda, I have shown that it is both feasible and relatively straightforward to automatically determine the necessary paths and reserve them appropriately on behalf of un-modified applications running in virtual execution environments. We showed that this considerably improved application performance (Section 9.2.2,[59]).

- **Integrated a passive network measurement tool with VNET:** Jointly with Ashish Gupta, Marcia Zangrilli, Bruce B. Lowekamp and Peter A. Dinda, I have shown that it is possible and feasible to monitor the performance of the underlying physical network by using the application's own traffic to automatically and cheaply probe it (Section 5).

I expect the following contributions from my thesis:

- **Answer to the question posed in the thesis statement** The most important contribution of my work will be an answer to the question posed in the thesis statement. I will know the feasibility of a single optimization metric for adapting a range of distributed applications executing in virtual execution environments.

- **Porting diverse distributed applications:** This will enable me to test my claim that adaptation using mechanisms made available by virtual execution environments are effective for a wide range of distributed applications. This will also provide myself and others an experimental testbed to test other characteristics and properties of adaptation mechanisms and the execution environment in general.

- **Extending work on optimization metrics:** I intend to come up with a single optimization scheme that will support a wide range of distributed applications. Currently application developers bear the burden of optimizing the application for its specific execution domain. This approach is not portable and very cumbersome. An adaptation scheme that is effective for a wide range of distributed applications will remove this burden and ease application development and deployment with no performance degradation in the worst case. Further, as discussed in Section 10.2 my approach of a generic optimization scheme will be more effective as distributed applications grow in size and adaptation control is distributed instead of being centralized.

- **Refining the adaptation formulation:** I will formulate the adaptation problem with respect to the single optimization metric that I come up with. In addition to being the first step in my search for an adaptation algorithm, it will be rigorous and abstract enough for many other adaptation problems to be mapped on to it.

- **Computational complexity characterization of the refined formulation:** I will characterize the computational complexity of the formulated problem in an attempt to better understand the solution space. If the problem turns out to be NP-hard, it could possibly be used to study the computational complexity of similar related optimization problems.

- **Approximate adaptation algorithms:** This will be one of my key contributions. Since it will be a solution to an abstract problem formulation, it has the potential to solve a wide range of real optimization problems.

- **A working adaptive system:** I will have a working adaptive virtual execution environment consisting of virtual machines connected via virtual networks. In addition to executing applications, it will also provide myself and others with a working setup to experiment and evaluate diverse research ideas related to adaptive virtual environments. My fellow graduate students and I will also integrate the different Virtuoso components into one single working system.

- **Extensive evaluation of the adaptation scheme:** I intend to carry out an extensive evaluation of the adaptive framework including comparing the developed heuristics against optimal solutions.This will help me understand the effectiveness of my approach in the context of a wide range of applications.

12. Related Work

In this section we discuss related work in wide-area distributed computing, virtual machine technology, virtual networks, adaptive overlay networks, virtual machine migration, measurement and inference, adaptation mechanisms and control, network reservations and network optimization problems.

12.1 Wide-area distributed computing

Recently there has been a great deal of interest in wide area distributed computing, primarily due to the substantial increase in commodity computer and network performance. This has allowed computational resources geographically distributed under different administrative domains and connected via wide area networks to be harnessed thus providing the illusion of a single unified computing resource. This is most commonly known as Grid computing [31]. Globus provides software infrastructure and services required to construct a computational grid [30, 29].

Legion is an object-based meta-system developed at the University of Virginia [35]. It provides the software infrastructure for a system of heterogeneous, geographically distributed high-performance computers

to interact seamlessly. WebFlow is a computational extension of the "web" model that can act as a framework for wide-area distributed computing [1]. Its main design goal was to build a seamless framework for publishing and reusing computational modules on the web. NetSolve is a client/server application designed to solve computational science problems in a distributed environment [15]. It searches for computational resources on the network, chooses the best one available, solves the problem and returns the answer to the user. Condor is a high throughput computing environment that can deliver large amounts of processing capacity over long periods of time by harnessing large collections of distributively owned heterogeneous computing resources.

Much grid middleware and application software is quite complex. Recently, interest in using OS-level virtual machines as the abstraction for distributed computing has been growing [28, 50, 32, 40]. Our group made the first detailed case for grid computing on virtual machines [28]. Being able to package a working virtual machine image that contains the correct operating system, libraries, middleware, and application can make it much easier to deploy something new, using relatively simple middleware that knows only about virtual machines. We have been developing a middleware system, Virtuoso, for virtual machine grid computing [78]. Others have shown how to incorporate virtual machines into the emerging grid standards environment [53].

12.2 Virtual machine technology

My work builds on operating-system level virtual machines, of which there are essentially two kinds. Virtual machine monitors, such as VMware [95], IBM's VM [45], and Microsoft's Virtual Server [67] present an abstraction that is identical to a physical machine. For example, VMWare, which we use, provides the abstraction of an Intel IA32-based PC (including one or more processors, memory, IDE or SCSI disk controllers, disks, network interface cards, video card, BIOS, etc.) On top of this abstraction, almost any existing PC operating system and its applications can be installed and run. The overhead of this emulation can be made to be quite low [83, 28]. Our work is also applicable to virtual server technology such as UML [20], Ensim [24], Denali [96], and Virtuozzo [94]. Here, existing operating systems are extended to provide a notion of server id (or protection domain) along with process id. Each OS call is then evaluated in the context of the server id of the calling process, giving the illusion that the processes associated with a particular server id are the only processes in the OS and providing root privileges that are effective only within that protection domain. In both cases, the virtual machine has the illusion of having network adaptors that it can use as it sees fit, which is the essential requirement of our work. VNET, without modification, has been successfully used with User Mode Linux [20] and the VServer extension to Linux [61].

12.3 Virtual networks

The Stanford Collective is seeking to create a compute utility in which "virtual appliances" (virtual machines with task-specialized operating systems and applications that are intended to be easy to maintain) can be run in a trusted environment [72, 32]. Part of the Collective middleware is able to create "virtual appli-

ance networks" (VANs), which essentially tie a group of virtual appliances to an Ethernet VLAN. My work is similar in that I also, in effect, tie a group of virtual machines together as a LAN. However, my work differs in that the collective middleware attempts also to solve IP address and routing, while we remain completely at the link layer and push this administration problem back to the user's site. Another difference is that I target the wide area environment in which remote sites are not under our administrative control. Hence, we make the administrative requirements at the remote site extremely simple and focused almost entirely on the machine that will host the virtual machine. Finally, because the nature of the applications and networking hardware in grid computing tend to be different (parallel scientific applications running on clusters with very high speed wide area networks) from virtual appliances, the nature of the adaptation problems and the exploitation of resource reservations made possible by VNET are also different. However, I should point out that one adaptation mechanism that I have used, migration, has been extensively studied by the Collective group [74].

Perhaps closest to VNET is that of Purdue's SODA project, which aims to build a service-on-demand grid infrastructure based on virtual server technology [50] and virtual networking [51]. Similar to VANs in the Collective, the SODA virtual network, VIOLIN, allows for the dynamic setup of an arbitrary private link layer and network layer virtual network among virtual servers. In contrast, VNET works entirely at the link layer and with the more general virtual machine monitor model. Furthermore, our model has been much more strongly motivated by the need to deal with unfriendly administrative policies at remote sites and to perform adaptation and exploit resource reservations. I have conducted a detailed performance analysis for VNET such results currently do not exists, to the best of our knowledge, for VAN or VIOLIN.

VNET is a virtual private network (VPN [25, 34, 48]) that implements a virtual local area network (VLAN [46]) spread over a wide area using link layer tunneling [92].

12.4 Adaptive overlay networks

I have extended VNET to act as an adaptive overlay network [2, 14, 43, 49] for virtual machines as opposed to for specific applications. The adaptation problems introduced are in some ways generalizations (because we have control over machine location as well as the overlay topology and routing) of the problems encountered in the design of and routing on overlays [77]. Further, VNET allows us to modify the overlay topology among a user's VMs at will. A key difference between it and overlay work in the application layer multicast community [7, 10, 44] is that the VNET provides global control of the topology, which our adaptation algorithms currently (but not necessarily) assume.

12.5 Virtual machine migration

Virtuoso allows us to migrate a VM from one physical host to another. Much work exists that demonstrates that fast migration of VMs running commodity applications and operating systems is possible [70, 73, 57]. Migration times down to 5 seconds have been reported [57]. As migration times decrease, the rate of adaptation we can support and our work's relevance increases. Note that while process migration and remote

execution has a long history [82, 23, 68, 91, 101], to use these facilities, we must modify or relink the application and/or use a particular OS. Neither is the case with VM migration. Although Virtuoso supports plug-in migration schemes, of which we have implemented copy using SSH, synchronization using rsync [93], and migration by transferring redo logs in a versioning file system [17]. In my work up till now, I have used rsync. I have still not decided as to the migration mechanism that I will use in my thesis.

12.6 Measurement and inference

At a very high level we adapt the application to the network and in the case of network reservations adapt the network to the application. In either case we need to have some means of inferring the application demands and measuring the underlying network.

I use the Virtual Topology and Traffic Inference Framework (VTTIF), developed by my fellow graduate student, Ashish Gupta [37]. VTTIF integrates with VNET to automatically infer the dynamic topology and traffic load of applications running inside the VMs in the Virtuoso system. I also intend to leverage well known mechanisms to measure available compute rate of the hosts [22, 99].

There is abundant work that suggests that underlying network measurements can be accomplished within or without the virtual network using both active [75, 100] and passive techniques [102, 64, 76]. In joint work with Ashish Gupta, Marcia Zangrilli, Bruce B. Lowekamp and Peter Dinda, I have shown that the naturally occurring traffic of an existing, unmodified application running in VMs can be used to measure the underlying physical network [38].

12.7 Adaptation mechanisms and control

An application running in some distributed computing environment must adapt to the (dynamically changing) available computational and networking resources to achieve stable high performance. Over the years there have been numerous attempts at adaptation in different settings such as load balancing in networks of shared processors [41, 98, 19], solutions to workflow problems, component placement problems and support for heavyweight applications in computational grids [11, 54, 62], adaptation, load balancing and fault tolerance in message passing and parallel processing systems spread over heterogeneous resources [79, 63, 3, 36], distributed mobile applications [69], automated runtime tuning systems [89] and extensions to commercial standards such as QuoIN/CORBA [104].

Despite these efforts adaptation and control mechanisms are not common in today's distributed computing environments as most of the approaches are very application-specific and require considerable user or developer effort. We have shown that adaptation using the low-level, *application-independent* adaptation mechanisms made possible by virtual machines interconnected with a virtual network is highly effective [85, 87, 59]. Furthermore, our adaptation mechanisms can be controlled automatically *without developer or user help.*

12.8 Network reservations

Much work has been done on simulating distributed applications and their communication behavior. Tools such as GridSim [84], SimGrid [16], and Prophesy [90] were developed by the grid community to model an application's behavior with the goal of understanding its computational and communication requirements. Using these models network reservations can be made before the application starts, using simulation results as predictors for network traffic requirements. Our system provides a true *run-time* reservation service that does not require any application simulations. Our system also alleviates the requirement that the user explicitly request advance reservations on behalf of the application.

Run-time adaptation of optical networks to ISP level traffic has been previously demonstrated [33]. Our work takes place at the opposite end of the spectrum; we measure and adapt for individual applications. The other work also treats the optical network as a closed topology, most often seen in the backbone infrastructure of large ISPs. The network topology was modified to reach an optimized state by measuring flow characteristics over the entire ISP. Our project complements this work because we simply make reservations on an optical network and do not care about the physical topology, so long as our bandwidth and latency requirements are met, while their work demonstrates a method of optimizing the physical topology to better meet collective demands.

Advance reservations [81] can be incorporated into optical and other kinds of networks to enhance application and network performance. VRESERVE can easily coexist with advance reservations because on-demand reservation requests are a special case of advance reservations [26]. An early version of VRESERVE specifically targeted operation with deferred reservation requests. While VRESERVE is able to accept deferred reservations, it is unable to make advance reservation decisions as it would have to predict, not just measure, application demands, a service that we have not yet developed

To the best of our knowledge no previous work exists that demonstrates on-demand run-time reservations for unmodified applications. Our system alleviates the need for application developers and/or users to directly interface with the reservation system. Reservation requests are made at run time and are dependent on the applications current communication requirements.

12.9 Network optimization problems

I have formulated the generic adaptation problem in virtual environments that aims to maximize the sum of the residual bottleneck bandwidths over all the mapped paths. A lot of related work exists in optimizing network flows. The closest work to mine is the unsplittable-flow problem (UFP) [55]. One of the motivations for formulating UFP is to address the problem of allocating bandwidth for traffic with different bandwidth requirements in heterogeneous networks. The UFP is MAXSNP-hard [39]. Approximation algorithms for the UFP and related problems have been presented in several prior works [55, 56, 39, 8]. Kleinberg [55] provides a comprehensive background on these problems. Baveja and Srinivasan [8] provided a $O(\sqrt{m})$, where m is the number of edges in the graph, approximate algorithm by an LP-based algorithm. Azar and Regev [4] gave a simpler, combinatorial algorithm with the same approximation guarantee. Kolliopoulos

and Stein [56] presented the first nontrivial approximation, $O(logm\sqrt{m})$, for a more general version of the UFP in which each request has an associated profit p_i and the goal is to maximize the total profit of accepted requests (UFP with profits). Further, my adaptation problem also has a strong connection to parallel task graph mapping problems [12, 58]. To the best of my knowledge no prior theoretical works exists that includes both the mapping and network flow components.

References

[1] AKARSU, E., FOX, G. C., FURMANSKI, W., AND HAUPT, T. Webflow: high-level programming environment and visual authoring toolkit for high performance distributed computing. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)* (Washington, DC, USA, 1998), IEEE Computer Society, pp. 1–7.

[2] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)* (2001).

[3] ARABE, J., BEGUELIN, A., LOWEKAMP, B., E. SELIGMAN, M. S., AND STEPHAN, P. Dome: Parallel programming in a heterogeneous multi-user environment. Tech. Rep. CMU-CS-95-137, Carnegie Mellon University, School of Computer Science, April 1995.

[4] AZAR, Y., AND REGEV, O. Strongly polynomial algorithms for the unsplittable flow problem. In *Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization* (London, UK, 2001), Springer-Verlag, pp. 15–29.

[5] BAILEY, D. H., BARSZCZ, E., BARTON, J. T., BROWNING, D. S., CARTER, R. L., DAGUM, D., FATOOHI, R. A., FREDERICKSON, P. O., LASINSKI, T. A., SCHREIBER, R. S., SIMON, H. D., VENKATAKRISHNAN, V., AND WEERATUNGA, S. K. The nas parallel benchmarks. *The International Journal of Supercomputer Applications 5*, 3 (Fall 1991), 63–73.

[6] BAKER, M., BUYYA, R., AND LAFORENZA, D. Grids and grid technologies for wide-area distributed computing. *Journal of Software Practice and Experience 32*, 15 (2002), 1437–1466.

[7] BANERJEE, S., LEE, S., BHATTACHARJEE, B., AND SRINIVASAN, A. Resilient multicast using overlays. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (June 2003).

[8] BAVEJA, A., AND SRINIVASAN, A. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research 25*, 2 (2000), 255–280.

[9] BERGER, M. J., AND COLELLA, P. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics 82*, 1 (1989), 64–84.

[10] BIRRER, S., AND BUSTAMANTE, F. Nemo: Resilient peer-to-peer multicast without the cost. In *Proceedings of the 12th Annual Multimedia Computing and Networking Conference* (January 2005).

[11] BLYTHE, J., DEELMAN, E., GIL, Y., KESSELMAN, C., AGARWAL, A., MEHTA, G., AND VAHI, K. The role of planning in grid computing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (2003).

[12] BOLLINGER, S., AND MIDKIFF, S. Heuristic techniques for processor and link assignment in multicomputers. *IEEE Transactions on Computers 40*, 3 (March 1991).

[13] BRYAN, G. L., ABEL, T., AND NORMAN, M. L. Achieving extreme resolution in numerical cosmology using adaptive mesh refinement: resolving primordial star formation. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)* (New York, NY, USA, 2001), ACM Press, pp. 13–13.

[14] CAMPBELL, A., KOUNAVIS, M., VILLELA, D., VICENTE, J., MEER, H. D., MIKI, K., AND KALAICHELVAN, K. Spawning networks. *IEEE Network* (July/August 1999), 16–29.

[15] CASANOVA, H., AND DONGARRA, J. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing 11*, 3 (Fall 1997), 212–223.

[16] CASANOVA, H., LEGRAND, A., AND MARCHAL, L. Scheduling distributed applications: the SimGrid simulation framework. In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)* (May 2003).

[17] CORNELL, B., DINDA, P., AND BUSTAMANTE, F. Wayback: A user-level versioning file system for linux. In *Proceedings of USENIX 2004 (Freenix Track)* (July 2004).

[18] CPLEX OPTIMIZATION INC. *Using the CPLEX callable library and the CPLEX mixed integer library*, 1993.

[19] CYBENKO, G. Dynamic load balancing for distributed shared memory multiprocessors. *Journal of Parallel and Distributed Computing 7*, 2 (October 1989), 279–301.

[20] DIKE, J. A user-mode port of the linux kernel. In *Proceedings of the USENIX Annual Linux Showcase and Conference* (Atlanta, GA, October 2000).

[21] DILGER, A., FLIERL, J., BEGG, L., GROVE, M., AND DISPOT, F. The PVM Patch for POV-Ray. Available at http://pvmpov.sourceforge.net.

[22] DINDA, P. A. Online prediction of the running time of tasks. *Cluster Computing 5*, 3 (2002). Earlier version appears in HPDC 2001. Summary in SIGMETRICS 2001.

[23] DOUGLIS, F., AND OUSTERHOUT, J. Process migration in the Sprite operating system. In *Proceedings of the 7th International Conference on Distributed Computing Systems (ICDCS)* (September 1987).

[24] ENSIM CORPORATION. http://www.ensim.com.

[25] FERGUSON, P., AND HUSTON, G. What is a vpn? Tech. rep., Cisco Systems, March 1998.

[26] FIGUEIRA, S., KAUSHIK, N., NAIKSATAM, S., CHIAPPARIC, S. A., AND BHATNAGAR, N. Advanced reservation of lightpaths in optical-network based grids. In *Proceedings of ICST/IEEE Gridnets* (October 2004).

[27] FIGUEIREDO, R., DINDA, P., AND FORTES, J. Resource virtualization renaissance. *IEEE Computer Special Issue On Resource Virtualization 38*, 5 (2005), 28–31.

[28] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003).

[29] FOSTER, I. Globus web page. http://www.mcs.anl.gov/globus.

[30] FOSTER, I., AND KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing 11*, 2 (Summer 1997), 115–128.

[31] FOSTER, I., KESSELMAN, C., AND TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications 15* (2001).

[32] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)* (October 2003).

[33] GENCATA, A., AND MUKHERJEE, B. Virtual-topology adaptation for WDM mesh networks under dynamic traffic. *IEEE/ACM Transactions on Networking 11*, 2 (2003), 236–247.

[34] GLEESON, B., LIN, A., HEINANEN, J., ARMITAGE, G., AND MALIS, A. A framework for IP-based virtual private networks. Tech. Rep. RFC 2764, Internet Engineering Taskforce, February 2000.

[35] GRIMSHAW, A., WULF, W., AND THE LEGION TEAM. The legion vision of a worldwide virtual computer. *Communications of the ACM 40*, 1 (1997).

[36] GRIMSHAW, A. S., STRAYER, W. T., AND P.NARAYAN. Dynamic object-oriented parallel processing. *IEEE Parallel and Distributed Technology: Systems and Applications*, 5 (May 1993), 33–47.

[37] GUPTA, A., AND DINDA, P. A. Inferring the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)* (June 2004).

[38] GUPTA, A., ZANGRILLI, M., SUNDARARAJ, A. I., DINDA, P., AND LOWEKAMP, B. Free network measurement for adaptive virtualized distributed computing. Tech. Rep. NWU-CS-05-13, Northwestern University, June 2005.

[39] GURUSWAMI, V., KHANNA, S., RAJARAMAN, R., SHEPHERD, B., AND YANNAKAKIS, M. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences 67*, 3 (2003), 473–496.

[40] HAND, S., HARRIS, T., KOTSOVINOS, E., AND PRATT, I. Controlling the xenoserver open platform. In *Proceedings of OPENARCH 2003* (April 2003).

[41] HARCHOL-BALTER, M., AND DOWNEY, A. B. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of ACM SIGMETRICS '96* (May 1996), pp. 13–24.

[42] HAROLD W. CAIN, RAVI RAJWAR, M. M., AND LIPASTI, M. H. An architectural evaluation of Java TPC-W. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture* (January 2001).

[43] HUA CHU, Y., RAO, S., SHESHAN, S., AND ZHANG, H. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM* (2001).

[44] HUA CHU, Y., RAO, S., AND ZHANG, H. A case for end-system multicast. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (2000), pp. 1–12.

[45] IBM CORPORATION. White paper: S/390 virtual image facility for linux, guide and reference. GC24-5930-03, Feb 2001.

[46] IEEE 802.1Q WORKING GROUP. 802.1q: Virtual lans. Tech. rep., IEEE, 2001.

[47] INTERNATIONAL CENTER FOR ADVANCED INTERNET RESEARCH. http://www.icair.org/omninet/.

[48] ITALIANO, G., RASTOGI, R., AND YENER, B. Restoration algorithms for virtual private networks in the hose model. In *Procedings of IEEE INFOCOM* (June 2002).

[49] JANNOTTI, J., GIFFORD, D., JOHNSON, K., KAASHOEK, M., AND JR., J. O. Overcast: Reliable multicasting with an overlay network. In *Proceedings of OSDI 2000* (October 2000).

[50] JIANG, X., AND XU, D. Soda: A service-on-demand architecture for application service hosting platforms. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC 2003)* (June 2003), pp. 174–183.

[51] JIANG, X., AND XU, D. Violin: Virtual internetworking on overlay infrastructure. Tech. Rep. CSD TR 03-027, Department of Computer Sciences, Purdue University, July 2003.

[52] KARP, R. *Compexity of Computer Computations*. Miller, R.E. and Thatcher, J.W. (Eds.). Plenum Press, New York, 1972, ch. Reducibility among combinatorial problems, pp. 85–103.

[53] KEAHEY, K., DOERING, K., AND FOSTER, I. From sandbox to playground: Dynamic virtual environments in the grid. In *Proceedings of the 5th International Workshop on Grid Computing* (November 2004).

[54] KICHKAYLO, T., AND KARAMCHETI, V. Optimal resource-aware deployment planning for component-based distributed applications. In *Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC)* (June 2004), pp. 150–159.

[55] KLEINBERG, J. *Approximation Algorithms for Disjoint Paths Problems*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1996.

[56] KOLLIOPOULOS, S. G., AND STEIN, C. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In *Proceedings of the 6th International IPCO Conference on Integer Programming and Combinatorial Optimization* (London, UK, 1998), Springer-Verlag, pp. 153–168.

[57] KOZUCH, M., SATYANARAYANAN, M., BRESSOUD, T., AND KE, Y. Efficient state transfer for Internet suspend/resume. Tech. Rep. IRP-TR-02-03, Intel Research Laboratory at Pittburgh, May 2002.

[58] KWONG, K., AND ISHFAQ, A. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing 59*, 3 (1999), 381–422.

[59] LANGE, J. R., SUNDARARAJ, A. I., AND DINDA, P. A. Automatic dynamic run-time optical network reservations. In *Proceedings of the Fourteenth International Symposium on High Performance Distributed Computing (HPDC)* (July 2005).

[60] LIN, B., AND DINDA, P. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of ACM/IEEE SC 2005 (Supercomputing)* (November 2005).

[61] LINUX VSERVER PROJECT. http://www.linux-vserver.org.

[62] LOPEZ, J., AND O'HALLARON, D. Support for interactive heavyweight services. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing HPDC* (2001).

[63] LOWEKAMP, B., AND BEGUELIN, A. Eco: Efficient collective operations for communication on heterogeneous networks. In *Proceedings of the 10th International Parallel Processing Symposium* (1996), pp. 399–406.

[64] LOWEKAMP, B., O'HALLARON, D., AND GROSS, T. Direct queries for discovering network resource properties in a distributed environment. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC99)* (August 1999), pp. 38–46.

[65] MAMBRETTI, J., WEINBERGER, J., CHEN, J., BACON, E., YEH, F., LILLETHUN, D., GROSSMAN, B., GU, Y., AND MAZZUCO, M. The photonic terastream: Enabling next generation applications through intelligent optical networking at iGRID2002. *Future Generation Computer Systems 19*, 6 (August 2003), 897–908.

[66] MCCANNE, S., AND JACOBSON, V. The BSD packet filter: A new architecture for user-level packet capture. In *Prodeedings of USENIX Annual Technical Conference* (1993), pp. 259–270.

[67] MICROSOFT CORPORATION. Virtual server beta release.

[68] MILOJICIC, D., DOUGLIS, F., PAINDAVEINE, Y., WHEELER, R., AND ZHOU, S. Process migration. *ACM Computing Surveys 32*, 3 (September 2000), 241–299.

[69] NOBLE, B. D., SATYANARAYANAN, M., NARAYANAN, D., TILTON, J. E., FLINN, J., AND WALKER, K. R. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)* (1997).

[70] OSMAN, S., SUBHRAVETI, D., SU, G., AND NIEH, J. The design and implementation of Zap: A system for migrating computing environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (December 2002).

[71] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER, A. Virtual network computing. *IEEE Internet Computing 2*, 1 (January/February 1998).

[72] SAPUNTZAKIS, C., BRUMLEY, D., CHANDRA, R., ZELDOVICH, N., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th Large Installation Systems Administration Conference (LISA 2003)* (October 2003).

[73] SAPUNTZAKIS, C., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M., AND ROSENBLUM, M. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (December 2002).

[74] SAPUNTZAKIS, C. P., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M. S., AND ROSENBLUM:, M. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)* (December 2002).

[75] SAVAGE, S. Sting: A TCP-based network measurement tool. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems* (1999).

[76] SESHAN, S., STEMM, M., AND KATZ, R. H. SPAND: Shared passive network performance discovery. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and System (USITS)* (97).

[77] SHI, S., AND TURNER, J. Routing in Overlay Multicast Networks. In *Proceedings of IEEE INFOCOM 2002* (June 2002).

[78] SHOYKHET, A., LANGE, J., AND DINDA, P. Virtuoso: A system for virtual machine marketplaces. Tech. Rep. NWU-CS-04-39, Department of Computer Science, Northwestern University, July 2004.

[79] SIEGELL, B., AND STEENKISTE, P. Automatic generation of parallel programs with dynamic load balancing. In *Proceedings of the Third International Symposium on High-Performance Distributed Computing (HPDC)* (August 1994), pp. 166–175.

[80] SMITH, W. D. TPC-W: benchmarking an ecommerce solution. Tech. rep., Intel Corporation.

[81] SNELL, Q., CLEMENT, M., JACKSON, D., AND GREGORY, C. The performance impact of advance reservation meta-scheduling. In *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)* (2000), pp. 137–153.

[82] STEENSGAARD, B., AND JUL, E. Object and native code process mobility among heterogeneous computers. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (December 1995), ACM.

[83] SUGERMAN, J., VENKITACHALAN, G., AND LIM, B.-H. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference* (June 2001).

[84] SULISTIO, A., PODUVALY, G., BUYYA, R., AND THAM, C.-K. Constructing a grid simulation with differentiated network servi ce using GridSim. Tech. Rep. GRIDS-TR-2004-13, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, December 2004.

[85] SUNDARARAJ, A., GUPTA, A., AND DINDA, P. Dynamic topology adaptation of virtual networks of virtual machines. In *Proceedings of the Seventh Workshop on Languages, Compilers and Run-time Support for Scalable Systems (LCR)* (November 2004).

[86] SUNDARARAJ, A. I., AND DINDA, P. A. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd USENIX Virtual Machine Research and Technology Symposium (VM 04)* (May 2004).

[87] SUNDARARAJ, A. I., GUPTA, A., AND DINDA, P. A. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the Fourteenth International Symposium on High Performance Distributed Computing (HPDC)* (July 2005).

[88] SUNDARARAJ, A. I., SANGHI, M., LANGE, J., AND DINDA, P. A. An optimization problem in adaptive virtual environments. *ACM SIGMETRICS Performance Evaluation Review 33*, 2 (2005).

[89] TAPUS, C., CHUNG, I.-H., AND HOLLINGSWORTH, J. Active harmony: Towards automated performance tuning. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing* (2002), pp. 1–11.

[90] TAYLOR, V., WU, X., GEISLER, J., LI, X., LAN, Z., TEVENS, R. S., HERELD, M., AND JUDSON, I. Prophesy: An infrastructure for analyzing and modeling the perfo rmance of parallel and distributed applications. In *Proceedings of the 9th International Symposium on High Performan ce Distributed Computing (HPDC)* (August 2000).

[91] THAIN, D., AND LIVNY, M. Bypass: A tool for building split execution systems. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)* (Pittsburgh, PA, August 2000).

[92] TOWNSLEY, W., VALENCIA, A., RUBENS, A., PALL, G., ZORN, G., AND PALTER, B. Layer two tunneling protocol "l2tp". Tech. Rep. RFC 2661, Internet Engineering Task Force, August 1999.

[93] TRIDGELL, A. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, 1999.

[94] VIRTUOZZO CORPORATION. http://www.swsoft.com.

[95] VMWARE CORPORATION. http://www.vmware.com.

[96] WHITAKER, A., SHAW, M., AND GRIBBLE, S. Scale and performance in the denali isolation kernel. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI 2002)* (December 2002).

[97] WHITE, S., LUND, A., AND SUNDERAM, V. S. Performance of the nas parallel benchmarks on pvm-based networks. *J. Parallel Distrib. Comput. 26*, 1 (1995), 61–71.

[98] WILLEBEEK-LEMAIR, M., AND REEVES, A. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems 4*, 9 (September 1993), 979–993.

[99] WOLSKI, R., SPRING, N., AND HAYES, J. Predicting the CPU availability of time-shared unix systems. In *Proceedings of the Eighth IEEE Symposium on High Performance Distributed Computing HPDC99* (August 1999), IEEE, pp. 105–112. Earlier version available as UCSD Technical Report Number CS98-602.

[100] WOLSKI, R., SPRING, N. T., AND HAYES, J. The network weather service: A distributed resource performance forecasting system. *Journal of Future Generation Computing Systems* (1999).

[101] ZANDY, V. C., MILLER, B. P., AND LIVNY, M. Process hijacking. In *Proceedings of the 8th IEEE Symposium on High Performance Distributed Computing (HPDC)* (Redondo Beach, CA, August 1999).

[102] ZANGRILLI, M., AND LOWEKAMP, B. Using passive traces of application traffic in a network monitoring system. In *of the Thirteenth IEEE International Symposium on High Performance Distributed Computing (HPDC 13)* (June 2004).

[103] ZANGRILLI, M., AND LOWEKAMP, B. B. Applying principles of active available bandwidth algorithms to passive tcp traces. In *Passive and Active Measurement Workshop (PAM 2005)* (March 2005), LNCS, pp. 333–336.

[104] ZINKY, J. A., BAKKEN, D. E., AND SCHANTZ, R. E. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems 3*, 1 (April 1997), 55–73.

Appendix

A Analysis of RPVEE and GAPVEE

A.1 Analysis of RPVEE

The decision version of RPVEE (RPVEED) is specified as:

Problem 4 (Decision version of Routing Problem In Virtual Execution Environments (RPVEED))
INPUT:

- *A directed graph* $G = (H, E)$
- *A function* bw *:* $E \rightarrow \mathbb{R}$
- *A set of ordered 3-tuples* $\mathcal{A} = \{(s_i, d_i, b_i) \mid s_i, d_i \in H;\ b_i \in \mathbb{R};\ i = 1, \ldots, m\}$

OUTPUT: $R : \mathcal{A} \rightarrow \mathcal{P}$ *such that*

- $\left(\mathrm{bw}_e\right) - \left(\sum_{e \in R(A_i)} b_i\right) \geq 0, \forall e \in E,$
- $\sum_{i=1}^{m} \left(\min_{e \in R(A_i)} \left\{\mathrm{rc}_e\right\}\right)$, *where* $\mathrm{rc}_e = (\mathrm{bw}_e - \sum_{e \in R(A_i)} b_i), \geq k, k \in \mathbb{R}$

Lemma 1 *If a polynomial-time solution exists for RPVEED, then a solution exists for any arbitrary instance of EDPP.*

Proof The existence of a polynomial time solution to the routing problem implies that $\sum_{i=1}^{m} \left(\min_{e \in R(A_i)} \left\{\mathrm{rc}_e\right\}\right)$, where $\mathrm{rc}_e = (\mathrm{bw}_e - \sum_{e \in R(A_i)} b_i), \geq k, k \in \mathbb{R}$. Since the contributions from any path can be either 0 or ϵ, it is implied that all 3-tuples are mapped such that only edges with weights $(i + \epsilon)$ are part of the paths, i.e. edges $\in E$ Further, each of these edges can be part of only a single path since all demands, d_i, are 1 and all the edge weights are < 2. This implies that each of the mapped paths are edge disjoint. This proves that the existence of a polynomial-time solution to the particular instance of our problem implies a solution to any arbitrary instance of the edge disjoint path problem.

Lemma 2 *If no polynomial-time solution exists to RPVEED then it implies that no polynomial-time solution exists for any arbitrary instance of EDPP.*

Proof We will prove this by contradiction. Lets assume a polynomial-time solution exists for the edge disjoint path problem, while no polynomial-time solution exists for the particular instance of the routing problem. The capacities for all the edges participating in the paths are $(1 + \epsilon)$ and each edge participates only in a single path. The residual bottleneck bandwidth for each mapped path will be ϵ. Since we would have successfully mapped k such paths, the sum of the residual bottleneck bandwidths for the paths would be $k \cdot \epsilon$. This implies that there exists a polynomial-time solution to the particular instance of the routing problem, thus completing the proof by contradiction.

Theorem 1 RPVEE is NP-hard.

Proof For reducing EDPP to an instance of RPVEED, construct a directed graph $G' = (V, E')$ where $\mathrm{bw}((u,v)) = 1 + \epsilon$ if $(u,v) \in E$ and $\mathrm{bw}((u,v)) = 1$ if $(u,v) \ni E$. Further for all $(s_i, t_i) \in S$, let $(s_i, d_i, 1) \in \mathcal{A}$. It is obvious that this reduction can be done in $O(n^2)$. Since we can reduce, in polynomial-time, a NP-complete problem, EDPP, to RPVEED and by Lemma 1 and Lemma 2, we have proved that the transformation works, we have proved RPVEED to be NP-hard. This proves that the optimization version of RPVEED, RPVEE, is NP-hard.

A.2 Analysis of GAPVEE

The NP-hardness for the problem is established by reduction from RPVEED, which has already been shown to be NP-complete. To prove the NP-hardness of GAPVEE we take any arbitrary instance of RPVEED and convert it to a particular instance of the decision version of GAPVEE, GAPVEED. We then show that RPVEED will have a "Yes" solution if and only if we have a solution to the GAPVEED. The decision version of GAPVEE (GAPVEED) is specified as follows:

Problem 5 (Decision version of Generic Adaptation Problem In Virtual Execution Environments (GAPVEED))

INPUT:

- *A directed graph* $G = (H, E)$
- *A function* bw *:* $E \rightarrow \mathbb{R}$
- *A function* lat *:* $E \rightarrow \mathbb{R}$
- *A function* $\mathrm{compute}$ *:* $H \rightarrow \mathbb{R}$
- *A function* size *:* $H \rightarrow \mathbb{R}$
- *A set,* $\mathrm{VM} = (\mathrm{vm}_1, \mathrm{vm}_2 \ldots \mathrm{vm}_n)$, $n \in \mathbb{N}$
- *A function* $\mathrm{vm\_compute}$ *:* $\mathrm{VM} \rightarrow \mathbb{R}$
- *A function* $\mathrm{vm\_size}$ *:* $\mathrm{VM} \rightarrow \mathbb{R}$
- *A set of ordered 4-tuples* $\mathcal{A} = \{(s_i, d_i, b_i, l_i) \mid s_i, d_i \in \mathrm{VM};\ b_i, l_i \in \mathbb{R};\ i = 1, \ldots, m\}$
- *A set of ordered pairs* $\mathcal{M} = \{(\mathrm{vm}_i, h_i) \mid \mathrm{vm}_i \in \mathrm{VM}, h_i \in H;\ i = 1, 2 \ldots r, r \leq n\}$

OUTPUT: $\mathrm{vmap} : VM \rightarrow H$ *and* $R : \mathcal{A} \rightarrow \mathcal{P}$ *such that*

- $\sum_{\mathrm{vmap(vm)}=h} \big(\mathrm{vm\_compute(vm)}\big) \leq \mathrm{compute(h)}, \forall\ h \in H$
- $\sum_{\mathrm{vmap(vm)}=h} \big(\mathrm{vm\_size(vm)}\big) \leq \mathrm{size}(h), \forall\ h \in H$
- $h_i = \mathrm{vmap}(\mathrm{vm}_i), \forall M_i = (\mathrm{vm}_i, h_i) \in \mathcal{M}$
- $\big(bw_e - \sum_{e \in R(A_i)} b_i\big) \geq 0, \forall e \in E$
- $\big(\sum_{e \in R(A_i)} \mathrm{lat}_e\big) \leq l_i, \forall e \in E$
- $\sum_{i=1}^{m} \big(\min_{e \in R(A_i)} \{\mathrm{rc}_e\}\big)$, *where* $\mathrm{rc}_e = (\mathrm{bw}_e - \sum_{e \in R(A_i)} b_i), \geq k, k \in \mathbb{R}$

Theorem 2 GAPVEE is NP-hard.

Proof For reducing RPVEED to an instance of GAPVEED, construct a directed graph $G = (H, E)$ where

- $\mathrm{lat}((u,v)) = 0 \forall (u,v) \in E$
- $\mathrm{compute}(h) = (n + \epsilon) \forall h \in H$
- $\mathrm{size}(h) = (n + \epsilon) \forall h \in H$

Also introduce a set, $\mathrm{VM} = (vm_1, \ldots vm_m)$, such that $\forall (s_i, d_i, b_i) \in \mathcal{A}$ (RPVEED), $s_i, d_i \in VM$. This would define the set of ordered pairs. Further $\forall \mathrm{vm} \in \mathrm{VM}$

- $\mathrm{vm\_compute(vm)} = 1$
- $\mathrm{vm\_size(vm)} = 1$

Finally for all $(s_i, d_i, b_i) \in \mathcal{A}$ (RPVEED), let $(s_i, d_i, b_i, 1) \in \mathcal{A}$ (GAPVEED). It is obvious that this reduction can be done in $O(n^2)$. Since we can reduce, in polynomial-time, a NP-complete problem, RPVEED, to GAPVEED and since it is trivially clear that a polynomial-time solution to RPVEED will exist if and only if a polynomial-time solution to GAPVEED exists, we have proved GAPVEED to be NP-hard. This proves that the optimization version of GAPVEED, GAPVEE is NP-hard.

B Greedy adaptation heuristics

B.1 Topology adaptation heuristic

Following is the topology adaptation algorithm, described in Section 9.3.2, in formal notation. The call $AdaptTopology(A, B, c)$, where A is the VM adjacency list, B is the VM to host mapping and c is the cost constraint, produces an ordered set S of fasts-path links that need to be added.

Procedure $AdaptTopology(A, B, c)$

1: $k \leftarrow 0$
2: **while** $count \neq SizeOf(A)$ **do** {loop invariant: $count \leq SizeOf(A)$}
3: {
$C[k, 0] \leftarrow B(A[k, 0])$
$C[k, 1] \leftarrow B(A[k, 1])$
$C[k, 2] \leftarrow A[k, 2]$
$k \leftarrow k + 1$
}
4: **end while**
5: $Quicksort(C)$
6: $count \leftarrow 0$
7: $i \leftarrow 0$
8: $k \leftarrow 0$
9: **while** $count \neq SizeOf(A)$ **do** {loop invariant: $count \leq SizeOf(A)$}
10: {
$S[k] \leftarrow LINK\ \ A[i, 0]\ \ A[i, 1]$
$k \leftarrow k + 1$
$i \leftarrow i + i$
$count \leftarrow count + 1$
}
11: **end while**
12: Return S

B.2 Migration adaptation heuristic

Following is the migration adaptation algorithm, described in Section 9.3.2, in formal notation. The call $MapVM(A, B, G)$, A is the VM adjacency list, B is the VNET daemon adjacency list and G is the VM to host mapping , produces a mapping from VMs to physical hosts.

Procedure $MapVM(A, B, G)$

1: $k \leftarrow 0$
2: **while** $count \neq SizeOf(A)$ **do** {loop invariant: $count \leq SizeOf(A)$}
3: {
$E[k, 0] \leftarrow G(A[k, 0])$
$E[k, 1] \leftarrow G(A[k, 1])$
$E[k, 2] \leftarrow A[k, 2]$
$k \leftarrow k + 1$
}
4: **end while**
5: $Quicksort(E)$
6: $Quicksort(B)$
7: $size \leftarrow SizeOf(E)$
8: $i \leftarrow 0$
9: $l \leftarrow 0$
10: Comment: C contains the VM nodes that have already been mapped, currently empty
11: Comment: D contains the host nodes that have already been mapped to, currently empty
12: Comment: T contains the new mapping from VMs to host nodes, currently empty
13: **while** $i \neq size$ **do** {loop invariant: $i \leq size$}
14: {
15: $m \leftarrow 0$
16: **if** $E[i, 0] \nexists C and E[i, 1] \nexists C$ **then**
17: $C[m] \leftarrow E[i, 0]$
18: $C[m + 1] \leftarrow E[i, 1]$
19: $m \leftarrow m + 2$
20: $j \leftarrow 0$
21: **while** $j \neq SizeOf(B)$ **do** {loop invariant: $j \leq SizeOf(B)$}
22: {
23: $n \leftarrow 0$
24: **if** $B[j, 0] \nexists D and B[j, 1] \nexists D$ **then**
25: $D[n] \leftarrow B[i, 0]$
26: $D[n + 1] \leftarrow B[i, 1]$
27: $n \leftarrow n + 1$
28: $T[l, 0] \leftarrow E[i, 0]$
29: $T[l, 1] \leftarrow B[j, 0]$
30: $l \leftarrow l + 1$
31: $T[l, 0] \leftarrow E[i, 1]$
32: $T[l, 1] \leftarrow B[j, 1]$
33: $l \leftarrow l + 1$
34: **end if**
}
35: **end while**
36: **end if**
$i \leftarrow i + i$
}
37: **end while**

```
38: i ← 0
39: while i ≠ size do {loop invariant: i ≤ size}
40:    {
41:    if E[i, 0]∉C then
42:       C[m] ← E[i, 0]
43:       m ← m + 1
44:       j ← 0
45:       found ← 0
46:       while j ≠ SizeOf(T)andfound == 0 do {loop invariant: j ≤ SizeOf(T)}
47:          {
48:          if E[i, 0] == T[j, 0] then
49:             temp ← T[j, 1]
50:             found ← 1
51:          end if
52:          if E[i, 0] == T[j, 1] then
53:             temp ← T[j, 0]
54:             found ← 1
55:          end if
             }
56:       end while
57:       T[l, 0] ← E[i, 0]
58:       T[l, 1] ← temp
59:       l ← l + 1
60:    end if
61:    if E[i, 1]∉C then
62:       C[m] ← E[i, 1]
63:       m ← m + 1
64:       j ← 0
65:       found ← 0
66:       while j ≠ SizeOf(T) and found == 0 do {loop invariant: j ≤ SizeOf(T)}
67:          {
68:          if E[i, 1] == T[j, 0] then
69:             temp ← T[j, 1]
70:             found ← 1
71:          end if
72:          if E[i, 1] == T[j, 1] then
73:             temp ← T[j, 0]
74:             found ← 1
75:          end if
             }
76:       end while
77:       T[l, 0] ← E[i, 1]
78:       T[l, 1] ← temp
79:       l ← l + 1
80:    end if
```

$count \leftarrow count + 1$
$\}$
end while
Return T