Dynamic Topology Adaptation of Virtual Networks of Virtual Machines

Ananth I. Sundararaj Ashish Gupta Peter A. Dinda
{ais, ashish, pdinda}@cs.northwestern.edu
Department of Computer Science, Northwestern University

ABSTRACT

Virtual machine grid computing greatly simplifies the use of widespread computing resources by lowering the level of abstraction, benefiting both resource providers and users. For the user, the Virtuoso middleware that we are developing closely emulates the existing process of buying, configuring and using machines. VNET, a component of Virtuoso, is a simple and efficient layer two virtual network tool that makes these virtual machines appear to be connected to the home network of the user, simplifying network management. Overlays like VNET have great potential as the mechanism for adaptation. Here, we describe our second generation VNET implementation, which includes support for arbitrary topologies and routing, application topology inference, and adaptive control of the overlay. We demonstrate that the performance of unmodified applications, in particular bulk synchronous parallel applications running inside the virtual machines and serviced by VNET, can be significantly (up to a factor of two) enhanced by adapting the VNET topology and forwarding rules on the fly based on intelligent application traffic inference methods. The adaptation scheme requires no knowledge or participation from the user or application developer.

1. INTRODUCTION

Virtual machines can greatly simplify grid and 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. 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 pre-installed versions of the correct operating system, libraries, middleware and applications can make the deployment of new software far simpler. We have made a detailed case for grid computing on virtual machines in a previous paper [2]. We are developing a middleware system, Virtuoso, for virtual machine grid computing [10].

Grid computing is intrinsically about using multiple sites, with different network management and security philosophies, often spread over the wide area [3]. 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. The impact of this variation is further exacerbated 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, we developed VNET [12], a simple layer 2 virtual network tool. 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, which also moves the network management problem from one network to another. Because the virtual network is a layer 2 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.[1]

An application running in some distributed computing environment, whether virtualized or not, must adapt to the (possibly changing) available computational and networking resources. Despite many efforts [15, 9, 6, 11, 1, 7], adaptation mechanisms and control have remained very application-specific. We are testing the proposition that adaptation using the application-independent capabilities made possible by virtual machines interconnected with a virtual network is effective. This paper provides initial evidence that the proposition is true. We are now determining the extent of applications for which it is true.

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. VNET

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, EIA-0224449, and a gift from VMWare. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

[1] virtuoso.cs.northwestern.edu

is in an ideal position to

1. measure the traffic load and application topology of the virtual machines
2. monitor the underlying network
3. adapt the application as measured in step 1. to the network as measured in step 2. by relocating virtual machines and modifying the virtual network topology and routing rules
4. take advantage of resource reservation mechanisms in the underlying network

Best of all, these services can be done on behalf of existing, unmodified applications and operating systems running in the virtual machines. A previous paper [12] laid out the argument and formalized the adaptation problem. Here we demonstrate adapting the virtual topology and routing rules to the measured application topology and traffic load.

In the next section, we describe Virtuoso, focusing on the components used for adaptation. We explain how VNET has been extended beyond its original implementation [12] to provide adaptation mechanisms, and we summarize our earlier results on the virtual topology and traffic inference framework (VTTIF) [4], which can infer an application's topology and traffic load matrix. In Section 3, we describe how we use VNET and VTTIF together to achieve automatic adaptation. We then report initial experimental results in Section 4 that show that our adaptation approach can be effective .

2. VIRTUOSO

We are developing middleware, Virtuoso, for virtual machine grid 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. Instead of a physical computer, the user receives a reference to the virtual machine which he 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. More details about the current Virtuoso implementation are available elsewhere [10].

The mechanisms in Virtuoso that are salient to this paper are VNET, our virtual networking system, and VTTIF, our application topology and traffic inference framework.

2.1 VNET

VNET is the part of our system that creates and maintains the networking illusion, that the user's virtual machines (VMs) are on the user's local area network. The specific mechanisms we use are packet filters, packet sockets, and VMware's [13] host-only networking interface. 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.

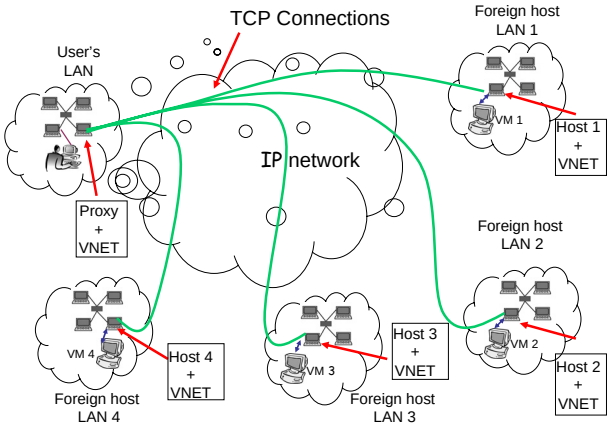

Figure 1: VNET startup topology.

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 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 reestablish them on failure.

The VNET daemons running on the hosts and Proxy open their virtual interfaces in promiscuous mode using Berkeley packet filters [8]. 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 connections) 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

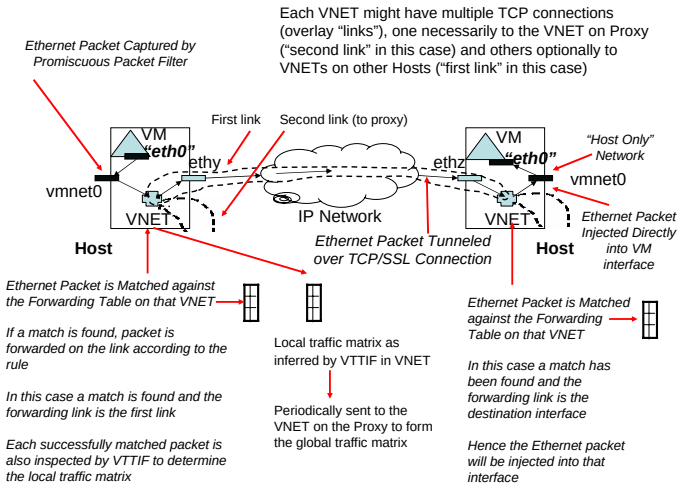

Figure 3: A VNET link.

match, the higher the priority value. For example, a rule matched with theänyqualifier 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 ïoneïs 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 [12], 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 paper, possible. Figure 8 shows a VNET configuration that has been dynamically adapted. The adaptation mechanisms are described in more detail in Sections 3 and 4.

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.

A language and its tools

Building on the primitives, we have developed a language for describing the topology and forwarding rules. Figure 4 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 and routing rules and a specified topology and 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 below).

2.2 VTTIF

The VTTIF component enables topology inference and traffic characterization for applications running inside the VMs in the Virtuoso system. 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 our earlier work [4], we 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.

Our system, VTTIF (virtual topology and traffic inference framework), works by examining each Ethernet packet that a VNET daemon receives from or delivers to a local VM.

⟨program⟩ ⟶ *BEGIN* ⟨host⟩ ⟨config⟩ *END*
⟨host⟩ ⟶ ⟨host⟩ *HOST* ⟨username⟩ *AT* ⟨machine⟩ ⟨port⟩ ⟨interface⟩ | ε
⟨config⟩ ⟶ ⟨config⟩ ⟨action⟩ ⟨link⟩ ⟨rules⟩ | ε
⟨action⟩ ⟶ *ADD* | *DELETE*
⟨link⟩ ⟶ ⟨link⟩ *LINK* ⟨machine⟩ ⟨machine⟩ | ε
⟨rules⟩ ⟶ ⟨rules⟩ *FORWARD* ⟨machine⟩ ⟨qualifier⟩ ⟨macaddress⟩ ⟨qualifier⟩ ⟨macaddress⟩ ⟨destination⟩ | ε
⟨destination⟩ ⟶ ⟨machine⟩ ⟨machine⟩ | ⟨interface⟩
⟨qualifier⟩ ⟶ *NOT* | *ANY* | ε
⟨macaddress⟩ ⟶ *Ethernet address such as 01 : 02 : 03 : 04 : 05 : 06*
⟨machine⟩ ⟶ *Machine name such as machine1*
⟨username⟩ ⟶ *User account on machine*
⟨port⟩ ⟶ *Port where VNET daemon runs*
⟨interface⟩ ⟶ *Ethernet interface such as eth0*
⟨at⟩ ⟶ *AT*

Figure 4: Grammar defining the language for describing VNET topology and forwarding rules.

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.

We found that we were able to accurately recover many common topologies from both synthetic and application benchmarks like the PVM-NAS benchmarks. For example, Figure 5 shows the topology inferred by VTTIF from the popular NAS benchmark Integer Sort [14] running on VMs. The thickness of each link reflects the intensity of communication along it. VTTIF adds very little overhead to VNET. Latency is essentially indistinguishable while throughput is effected on the order of 1%.

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 VTTIF can react, 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:

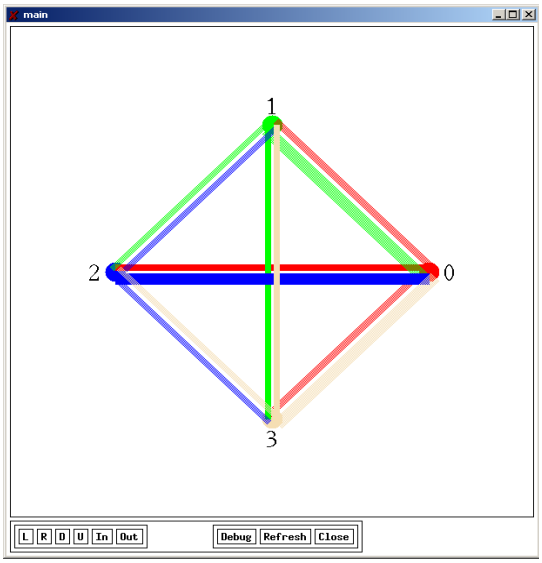

Figure 5: The NAS IS benchmark running on 4 VM hosts as inferred by VNET-VTTIF.

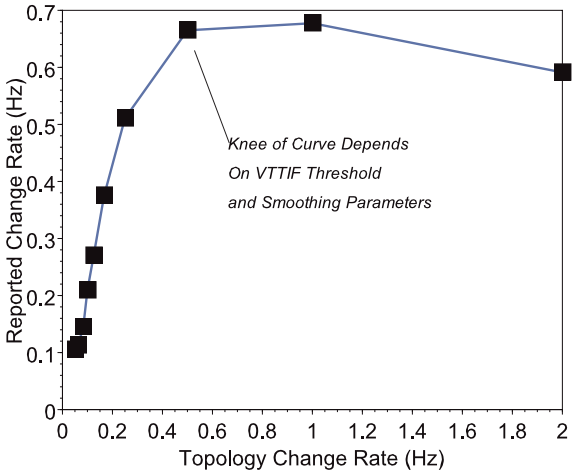

Figure 6: VTTIF is well damped.

- 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

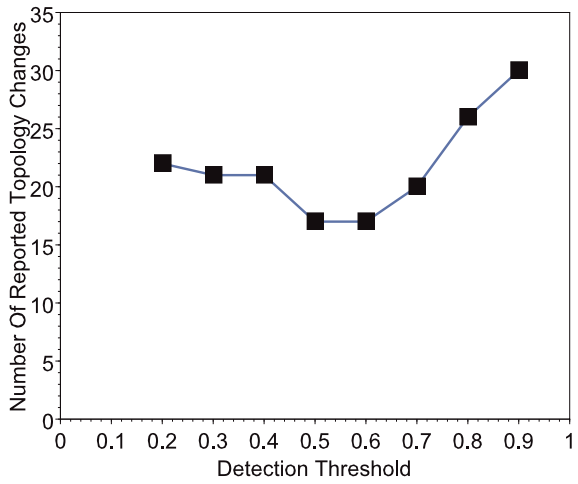

Figure 7: VTTIF is largely insensitive to the detection threshold.

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 now further topology changes. The best case settle time that we have measured on 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, we have designed VTTIF to stop reacting, settling into a topology that is effectively a union of all the topologies that are unfolding in the network. Figure 6 shows that VTTIF is indeed well damped. Here, we are using two separate topologies and switching rapidly between them. When this topology change rate exceeds VTTIF's rate, the reported change rate settles and declines. The knee of the curve depends on the choice of smoothing interval and update rate, with the best case being about 1 second. Essentially, up to this limit, the rate and interval set the knee according to the Nyquist criterion.

Within limits, VTTIF is largely insensitive to the choice of detection threshold, as shown in Figure 7. However, this parameter does determine the extent to which similar topologies can be distinguished.

3. ADAPTATION

For an application running in some distributed environment to achieve optimum performance, it must adapt to the (possibly changing) available computational and networking resources. Adaptation at the level of a collection of virtual machines connected by a virtual network, as provided by Virtuoso, presents tremendous opportunities and challenges. In the ideal case, such adaptation would mean that we could dynamically optimize, at run-time, the performance of existing, unmodified applications running on existing, unmodified operating systems without any user or programmer intervention. However, a number of challenges must be overcome before this vision transcends pollyanna-ism:

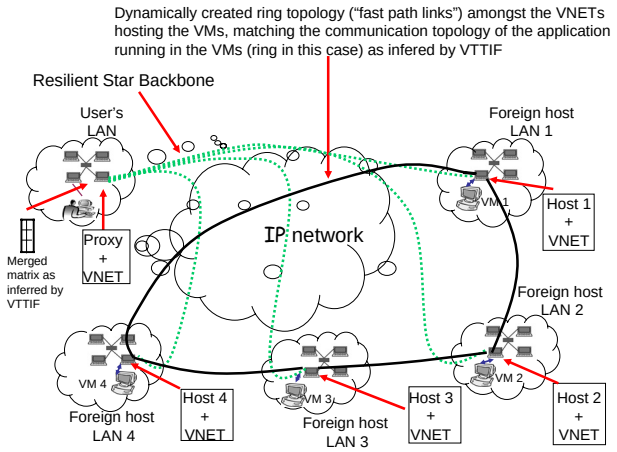

Figure 8: As the application progresses VNET adapts its overlay topology to match that of the application communication as inferred by VTTIF leading to an significant improvement in application performance, without any participation from the user.

- We must be able to adequately monitor the network and hosts from the VNET vantage point. Currently, we have no evidence on this.
- We must be able to adequately monitor the application. We have demonstrated that we can do so for BSP applications [4].
- We must be able to infer the goals of the applications (in terms of latency, throughput, etc). If this is infeasible, we must make the interface to the programmer or user to get this information as thin and easy to use as possible.
- We must show that the limited adaptation mechanisms available in the virtual environment level (topology changes, routing changes, virtual machine migration, and resource reservations) are adequate for adaptation.

Clearly, these challenges are interrelated, and they can probably be overcome for some subset of conceivable applications and not for others. We are currently trying to determine that subset. In the following, we demonstrate that the subset is not empty.

4. EXPERIMENTS

We focus on a specific instance of the challenges described in Section 3, looking at patterns, a synthetic benchmark that can be configured with most common communication patterns [4]. We try to minimize the running time of patterns, instantiated on different numbers of VMs, on different collections of hosts from different clusters and environments. Our adaption mechanism is to add links (and corresponding forwarding rules) to the VNET topology, guided by the topology inferred by VTTIF.

Figure 8 illustrates this dynamic adaptation. Here, patterns, configured with neighbor-exchange on a ring application topology of four VMs, starts executing with a VNET

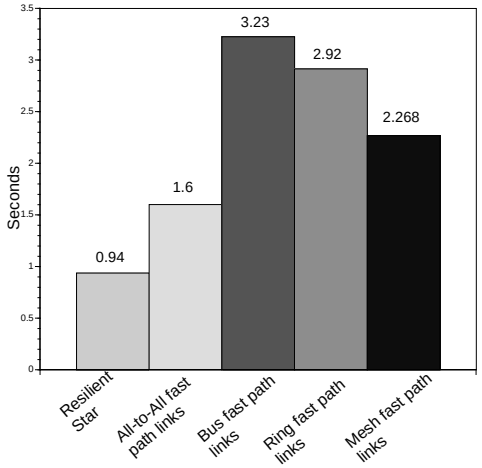

Figure 9: Time to set up the backbone star configuration and to add fast path links for different (inferred) topologies.

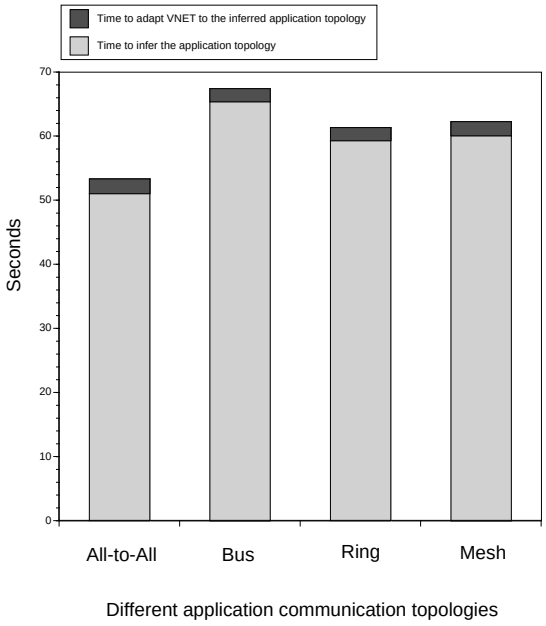

Figure 10: Time taken to infer the application topology, adapt VNET to it and the total application running time for different application topologies, for eight VMs on a single cluster.

star topology (dotted lines) centered on the Proxy. As patterns continues to execute, VTTIF infers that a ring topology exists. In response, VNET adds four links (dark lines) to form an overlay ring amongst the VNET daemons, thus matching the application's topology. If the time taken for inferring the application topology and adapting the VNET topology is a small fraction of the application lifetime, the VMs will spend most of their time executing with an efficient VNET topology, resulting in a lower running time.

The links that are added form the *fast path topology*, as they lead to faster communication between the application components. For example, VM1 can now communicate directly over a TCP connection to VM2, instead of redirecting its traffic to the VNET on the Proxy. It should be noted that the resilient star topology is maintained at all times. The fast path topology and its associated forwarding rules are modified as needed to improve performance.

4.1 Reaction time

Figure 9 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 amongst 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 longer (3.23 seconds), even though there are fewer links. This is because VNET does not use hierarchical routing. Eventually, migration of virtual machines will be another adaptation mechanism that we will leverage. Since VNET operates at layer 2, 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.

For the single cluster setup with 8 VMs, Figure 10 shows the time needed to infer and then adapt completely (add all inferred links) to different application topologies. Inferring the application topology takes the bulk of the time, on the order of 60 seconds, slightly varying for different patterns. The time taken to create the inferred fast path topology is a small portion of the time, on the order of 2 seconds.

Notice that the inference time is large because of the update rate, smoothing interval, and threshold settings chosen for VTTIF. Recall from Section 2.2 that VTTIF can react in as little as one second if configured to do so. Our results are conservative in this section.

Also note that while in this section we show a reaction to a single persistent topology, going from the Proxy star to the appropriate topology that the application exhibits over a long period, our adaptation system continually adapts to any topology changes that occur.

Finally, recall that VTTIF damps its reactions according to the smoothing interval, as described in Section 2.2. Because of this, once VNET determines how quickly it can change topology in the given physical environment, it can set VTTIF's smoothing interval accordingly. Because of the damping, VTTIF will then not ask VNET to adapt to a topol-

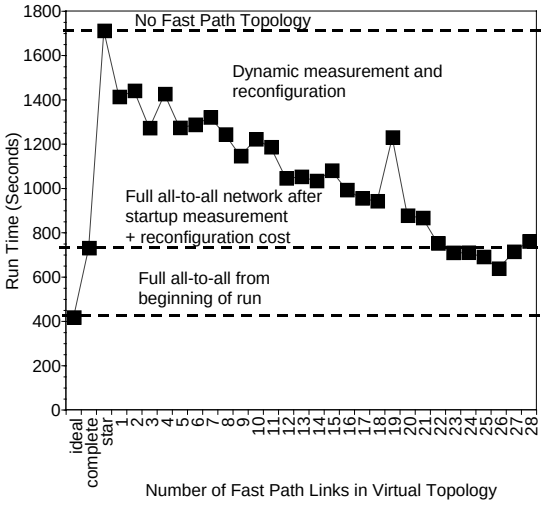

Figure 11: All-to-all topology with eight VMs, all on the same cluster.

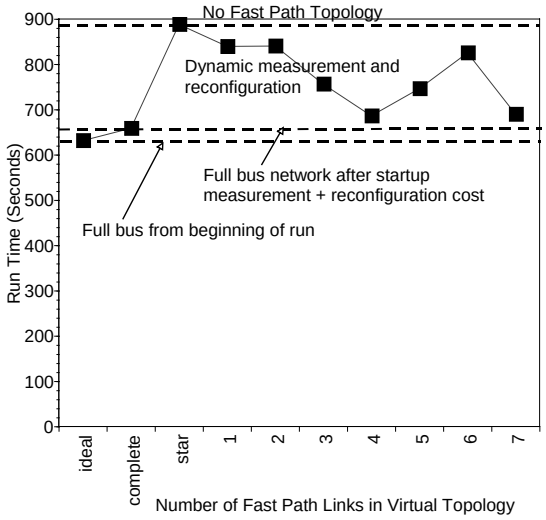

Figure 12: Bus topology with eight VMs, spread over two clusters over a MAN.

ogy that is changing too quickly. Instead, it will present a topology that is a union of the various topologies that are alternating on the network. Through this interaction, we make it impossible for our adaptation system to oscillate.

4.2 Benefits

We now study the benefits accrued as a function of the number of fast path links added. If we add k of the n inferred links, in decreasing order of traffic intensity, how much do we reduce the running time of patterns?

We repeated this experiment for the common topologies shown in Figure 10 for four and eight VM configurations running on four different environments. In addition to running all the VMs in a single cluster (Figure 11), as described above, we also split the VMs between two adjacent clusters (a second IBM e1350 as described above, connected by a 10 mbit connection via two firewalls and a switch), clusters separated by a metropolitan area campus network (the e1350 described above and a second cluster of dual 1 GHz P3 machines, with two firewalls in the path), and a wide area configuration with machines in our clusters, other IBM e1350s in DOT[2], and a machine at CMU.

Figure 11 gives an example for the single cluster configuration, here running an 8 VM all-to-all. Relying on the startup VNET configuration the application completes in about 1700 seconds, the running time comes down to 1400 seconds when the highest priority fast path link is added. It further reduces to about 1000 seconds when half (14) of the maximum possible fast path links (28) are added. The running time is reduced to about 700 seconds when all the fast path links inferred by VTTIF are added, an improvement of a factor of two.

Figure 12 illustrates the performance gain as a function of the number of fast path links for eight hosts in the two MAN-separated clusters. As before, each host runs a single VM. Note that although the P3 nodes are slower, patterns

[2]www.dotresearch.org

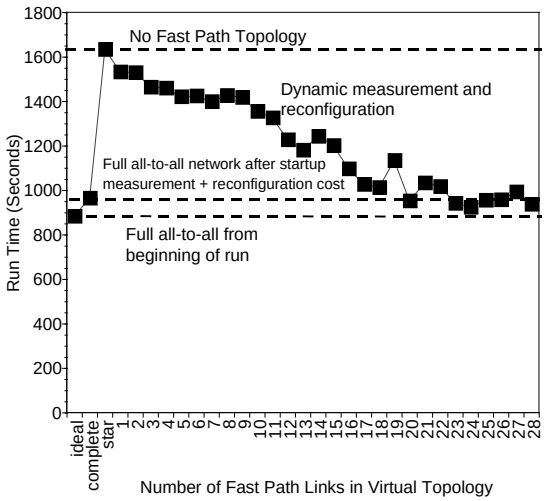

Figure 13: All-to-all topology with eight VMs, spread over a WAN.

is configured solely for communication, and the network capabilities of the P3 nodes and the Xeon nodes are identical. Here we show a bus communication topology. This is the worst performance that we have measured. Note that even here adaptation provides some gains.

Figure 13 shows performance for 8 VMs, all-to-all, in the WAN scenario, wherein the hosts are located on different networks spread over the WAN. Three hosts are located on a single cluster at Northwestern University, Evanston, two hosts are on another cluster located on the same campus, another host is located on a third MAN network and finally one host each is located at the University of Chicago, Chicago and Carnegie Mellon University (CMU), Pittsburgh. The Proxy and the user are located on a separate network on the Northwestern University campus. Again, we see a significant performance improvement as more and more fast path links are added.

Discussion

The results of this section clearly illustrate that it is possible to use our inference tool, VTTIF, the adaptation mechanisms of VNET, and a simple control algorithm to greatly increase the performance of existing, unmodified applications running in a VM environment like Virtuoso.

Our adaptation algorithm is currently centralized at the Proxy and uses global information. We are exploring the extent to which this limits scalability of our system. It is important to realize that it is the path of packets and the location of VMs that will determine the ultimate performance of an application in the common case. However, an application with a highly dynamic topology may strain a centralized adaptation algorithm. We are also exploring distributed adaptation algorithms and the ability for a user to introduce agents into the system that run his own chosen adaptation algorithms.

It is a common belief that lowering the level of abstraction increases performance while increasing complexity. In this particular case, the rule may not apply. Our abstraction for the user is identical to his existing model of a group of machines, but we can increase the performance he sees. In addition, it is our belief that lowering of the level of abstraction also makes adaptation much more straightforward to accomplish.

5. CONCLUSIONS

We have demonstrated the feasibility of adaptation at the level of a collection of virtual machines connected by a virtual network and shown that its benefits can be significant. Specifically, we can, at run-time, infer the communication topology of a BSP application executing in a set of VMs, and then change our virtual topology to partially or completely match, decreasing the application's execution time. No modifications to the application or its OS are needed, and our techniques place no requirements on the two other than they generate Ethernet packets. We are now studying the extent of applications for which our approach is effective, including examining non-parallel application benchmarks such as TPC-W [5]. We are also moving ahead to use other adaptation mechanisms, namely VM migration and the use of resource reservation mechanisms in the underlying network.

6. REFERENCES

[1] 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.

[2] 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).

[3] Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications 15* (2001).

[4] Gupta, A., and Dinda, P. A. Infering the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Policies for Parallel Program Processing(JSPPP 2004)* (June 2004).

[5] 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).

[6] 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* (2001).

[7] 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.

[8] McCanne, S., and Jacobson, V. The BSD packet filter: A new architecture for user-level packet capture. In *Prodeedings of USENIX 1993* (1993), pp. 259–270.

[9] 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* (1997). To Appear.

[10] 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.

[11] 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* (August 1994), pp. 166–175.

[12] 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 2004)* (May 2004).

[13] VMWare Corporation. http://www.vmware.com.

[14] White, S., Alund, A., and Sunderam, V. S. Performance of the NAS parallel benchmarks on PVM-Based networks. *Journal of Parallel and Distributed Computing 26*, 1 (1995), 61–71.

[15] 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.