Increasing Application Performance In Virtual Environments Through Run-time Inference and Adaptation

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

Abstract

Virtual machine distributed computing greatly simplifies the use of widespread computing resources by lowering the level of abstraction, benefiting both resource providers and users. Towards that end our Virtuoso middleware closely emulates the existing process of buying, configuring and using physical machines. Virtuoso's VNET component is a simple and efficient layer two virtual network tool that makes these virtual machines (VMs) appear to be physically connected to the home network of the user while simultaneously supporting arbitrary topologies and routing among them. Virtuoso's VTTIF component continually infers the communication behavior of the application running in a collection of VMs. The combination of overlays like VNET and inference frameworks like VTTIF has great potential to increase the performance, with no user or developer involvement, of existing, unmodified applications by adapting their virtual environments to the underlying computing infrastructure to best suit the applications. We show here how to use the continually inferred application topology and traffic to dynamically control three mechanisms of adaptation, VM migration, overlay topology, and forwarding to significantly increase the performance of two classes of applications, bulk synchronous parallel applications and transactional web ecommerce applications.

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 preinstalled versions of the correct operating system, libraries, middleware and applications can make the deployment of new software far simpler. We made the first detailed case for grid computing on virtual machines in a previous paper [10] and we have been developing a middleware system, Virtuoso, for virtual machine grid computing [37]. Others have shown how to incorporate virtual machines into the emerging grid standards environment [19]. An introduction to the state of the art in resource virtualization is also available [9].

Grid computing is intrinsically about using multiple sites, with different network management and security philosophies, often spread over the wide area [11]. 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 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 [41], 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. The first version of VNET is publicly available. In part, this paper reports on dramatically extended second version.

An application running in some distributed comput-

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, EIA-0224449, and gifts from VMware and Dell. 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), VMware, or Dell.

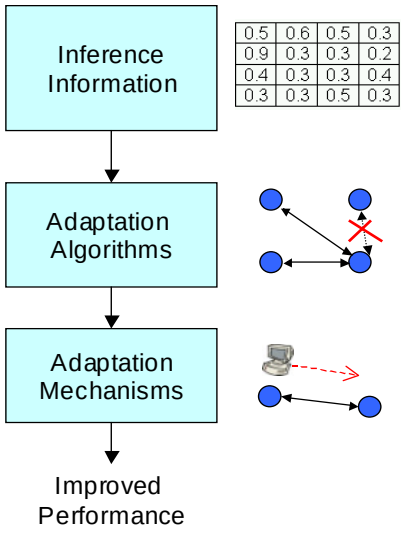

Figure 1. Application inference, adaptation algorithms, and adaptation mechanisms (overlay topology and VM migration) work together to increase application performance.

ing environment must adapt to the (dynamically changing) available computational and networking resources to achieve stable high performance. Nonetheless, despite many efforts [51, 31, 25, 38, 1, 26, 42, 12, 13, 16, 6, 47, 20, 4], adaptation mechanisms and control are not common on today's applications. The reason why is that they tend to be both very application-specific and require considerable user or developer effort. We claim that adaptation using the low-level, *application-independent* adaptation mechanisms made possible by virtual machines interconnected with a virtual network is highly effective. Furthermore, the mechanisms can be controlled automatically *without developer or user help.* This paper provides evidence for this claim.

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 is in an ideal position to

1. measure the traffic load and application topology of the virtual machines,
2. monitor the underlying network and its topology,
3. adapt the application as measured in step 1 to the network as measured in step 2, and
4. adapt the network to the application by taking advantage of resource reservation mechanisms.

This paper focuses on steps 1 and 3, as further illustrated in Figure 1. There is abundant work that suggests that step 2 can be accomplished within or without the virtual network using both active [35, 48] and passive techniques [50, 27, 36] and we have begun developing our own approach [15]. We are just beginning to work on step 4 [22, 23]

These services can be done on behalf of *existing, unmodified applications and operating systems* running in the virtual machines. One previous paper [41] laid out the argument and formalized the adaptation problem, while a second (workshop) paper [40] gave very preliminary results on automatic adaptation using one mechanism. Here, we demonstrate how to control three adaptation mechanisms provided by our system in response to the inferred communication behavior of the application running in a collection of virtual machines, and provide extensive evaluation.

We use the following three adaptation mechanisms:

- 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 [32, 34, 21]. Migration times down to 5 seconds have been reported [21]. 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 [39, 8, 30, 43, 49], 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 [2, 3, 18] 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, and topology and routing are completely separated, unlike in multicast systems.

2 Virtuoso

Virtuoso is a system 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. Virtuoso does rudimentary admission control of VMs, but the work described here additionally provides the ability for the system to adapt when the user cannot state his resource requirements, and the ability to support a mode of operation in which VMs and other processes compete for resources. In effect, the more competition, the cheaper the cost of admission. More details are available elsewhere [37].

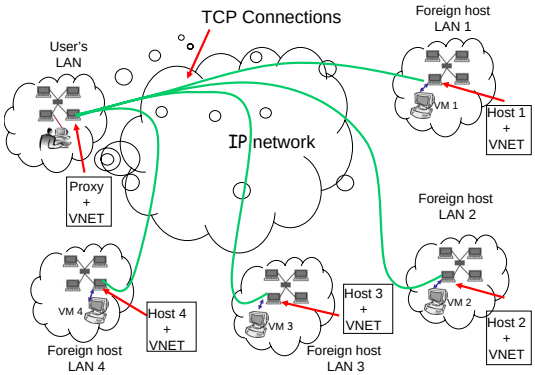

Figure 2. VNET startup topology.

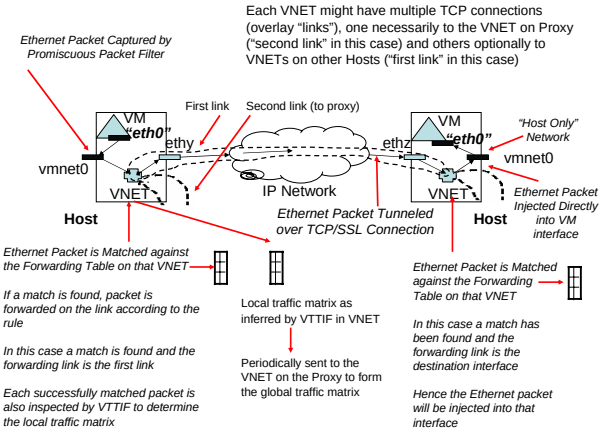

Figure 3. A VNET link.

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 [45] 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.

Although we use VMware as our virtual machine monitor (VMM), VNET can operate with any VMM that provides an externally visible representation of the virtual network interface. For example, VNET, without modification, has been successfully used with User Mode Linux [7] and the VServer extension to Linux [24].

Figure 2 shows a typical startup configuration of VNET for four hosts, each of which may support multiple VMs. Each of the VNET daemons 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. We would not be running a VM on any of these host machines if it were not possible in some way to communicate with them. This communication mechanism can be exploited to provide VNET connectivity for a remote VM. For example, if an SSH connection can be made to the host, VNET traffic can be tunneled over the SSH connection.

The VNET daemons running on the hosts and Proxy open their virtual interfaces in promiscuous mode using Berkeley packet filters [29]. Each packet captured from the interface or received on a link is matched against a forwarding table to determine where to send it, the possible choices being sending it over one of its outgoing links or writing it out to one of its local interfaces using libnet, which is built on packet sockets, available on both Unix and Windows.

Figure 3 illustrates the operation of a VNET link. Each successfully matched packet is also passed to VTTIF. 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 and his site administrator.

The star topology is simply the initial configuration. Additional links and forwarding rules can be added or removed at any time. In the case of migration, the VM seamlessly maintains its layer 2 and layer 3 network presence; neither MAC nor IP addresses change and the external network presence of the VM remains on the LAN of the Proxy. Figure 8 shows a VNET configuration that has been dynamically adapted to reflect a topology change.

A VNET client can query any VNET daemon for available network interfaces, links, and forwarding rules. It can add or remove overlay links and forwarding rules. The primitives generally execute in ∼20 ms, including client time. On initial startup VNET calculates an upper bound on the time taken to configure itself (or change topology). This number is used to determine sampling and smoothing intervals in VTTIF, as we describe below.

Building on the primitives, we have developed a language for describing the VM to host mapping, the topology, and its forwarding rules. A VNET overlay is usually managed using scripts that generate or parse descriptions in that language. We can

- Start up a collection of VNET daemons and establish an initial topology among them.
- Fetch and display the current topology and VM mappings.
- Fetch and display the route a packet will take between two Ethernet addresses.

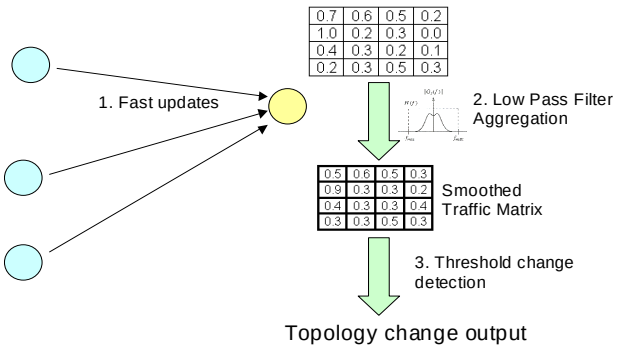

Figure 4. An overview of the dynamic topology inference mechanism in VTTIF.

- Compute the differences between the current topology, forwarding rules, and mappings and a specified topology, forwarding rules, and mappings.
- Reconfigure the topology, forwarding rules, and VM mappings to match a specified topology, forwarding rules, and mappings.
- Fetch and display the current application topology using VTTIF.

2.2 VTTIF

The Virtual Topology and Traffic Inference Framework integrates with VNET to automatically infer the dynamic topology and traffic load of applications running inside the VMs in the Virtuoso system. In our earlier work [14], we demonstrated that it is possible to successfully infer the behavior of a BSP application by observing the low level traffic sent and received by each VM in which it is running. Here we show how to smooth VTTIF's reactions so that adaptation decisions made on its output cannot lead to oscillation.

VTTIF works by examining each Ethernet packet that a VNET daemon receives from a local VM. VNET daemons collectively aggregate this information producing a global traffic matrix for all the VMs in the system. The application topology is then recovered from this matrix by applying normalization and pruning techniques [14]. Since the monitoring is done below the VM, it does not depend on the application or the operating system in any manner. VTTIF automatically reacts to interesting changes in traffic patterns and reports them, driving the adaptation process. Figure 4 illustrates VTTIF.

VTTIF can accurately recover 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 NAS benchmark Integer

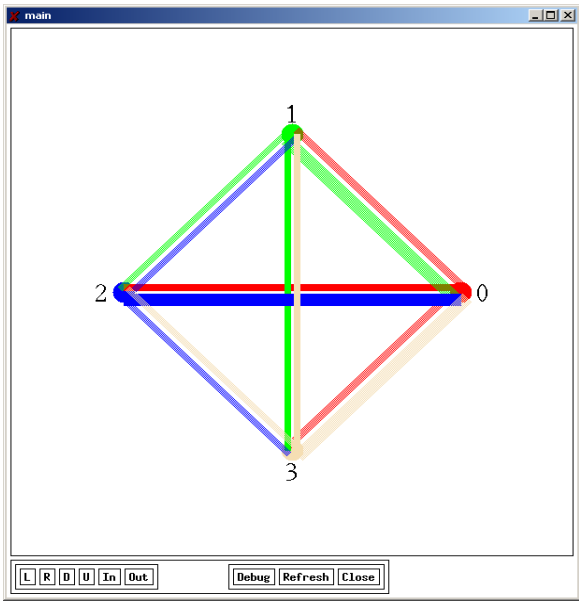

Figure 5. The NAS IS benchmark running on 4 VM hosts as inferred by VTTIF.

Sort [46] running on VMs. The thickness of each link reflects the intensity of communication along it. VTTIF adds little overhead to VNET. Latency is indistinguishable while throughput is affected by $\sim$1%.

Performance VTTIF runs continuously, updating its view of the topology and traffic load matrix among a collection of Ethernet addresses being supported by VNET. However, in the face of dynamic changes, 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? VTTIF also depends on certain configuration parameters which affect its decision whether the topology has changed. How sensitive is VTTIF to the choice of configuration parameters in its inference algorithm?

The reaction time of VTTIF depends on the rate of updates from the individual VNET daemons. A fast *update rate* imposes network overhead but allows a finer time granularity over which topology changes can be detected. In our current implementation, at the fastest, these updates arrive at a rate of 20 Hz. At the Proxy, VTTIF then aggregates the updates into a global traffic matrix. To provide a stable view of dynamic changes, it applies a low pass filter to the updates, aggregating the updates over a sliding window and basing its decisions upon this aggregated view.

Whether VTTIF reacts to an update by declaring that the topology has changed depends on the *smoothing interval*

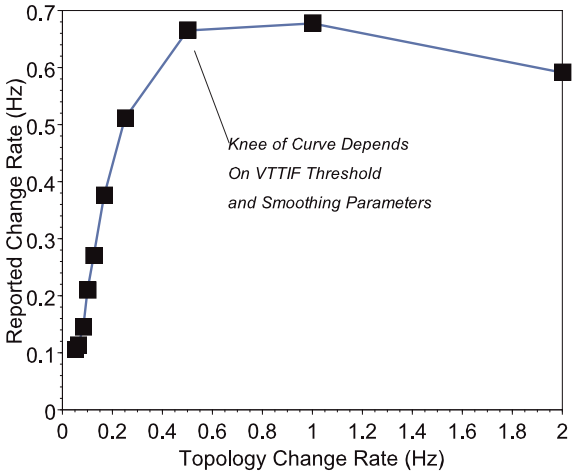

Figure 6. VTTIF is well damped.

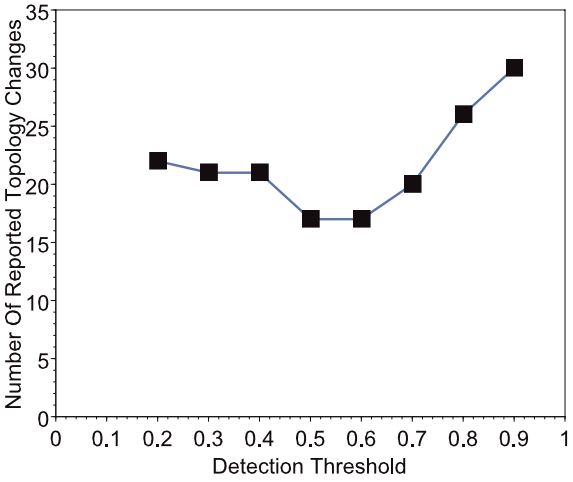

Figure 7. VTTIF is largely insensitive to the detection threshold.

and the *detection threshold*. The smoothing interval is the sliding window duration over which the updates are aggregated. This parameter depends on the adaptation time of VNET, which is measured at startup, and determines how long a change must persist before VTTIF notices. The detection threshold determines if the change in the aggregated global traffic matrix is large enough to declare a change in topology. 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 we have measured is one second, on par with the adaptation mechanisms.

Given an 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 a union of all the topologies that are unfolding in the network. Figure 6 shows the reaction rate of VTTIF as a function of the topology change rate and shows that it is indeed well damped. Here, we are using two separate topologies and switching rapidly between them. When this topology change rate exceeds VTTIF's configured 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 $\sim$1 second. Up to this limit, the rate and interval set the knee according to the Nyquist criterion.

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. Note that appropriate settings of the VTTIF parameters are determined by the *adaptation mechanisms*, not the *application*.

3 Adaptation and VADAPT

Virtuoso uses VTTIF to determine the communication behavior of the application running in a collection of VMs and can leverage the plethora of existing work on network monitoring ([28] is a good taxonomy) to determine the behavior of the underlying resources. The VNET component of Virtuoso provides the mechanisms needed to adapt the application to the network. Beyond this, what is needed is

- the measure of application performance, and
- the algorithms to control the adaptation mechanisms in response to the application and network behaviors.

Here the measure is the throughput of the application.

The adaptation control algorithms are implemented in the VADAPT component of Virtuoso. For a formalization of the adaptation control problem, please see our previous work [41]. 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 that maximizes the application throughput?"

VADAPT uses greedy heuristic algorithms to quickly answer this question when application information is available, and VM migration and topology/forwarding rule changes are the adaptation mechanisms.

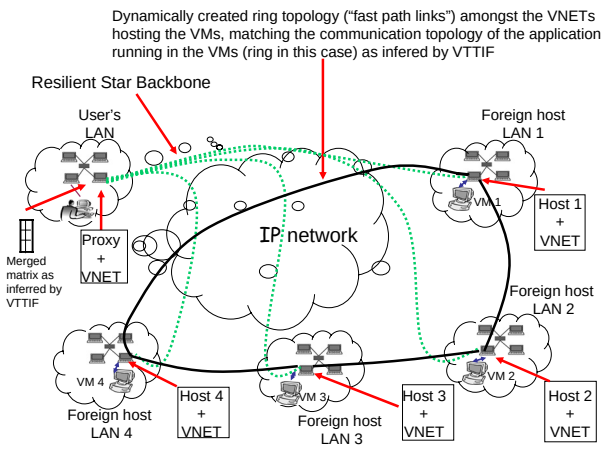

Figure 8. As the application progresses VNET adapts its overlay topology to match that of the application communication as inferred by VTTIF leading to an significant improvement in application performance, without any participation from the user.

3.1 Topology adaptation

VADAPT uses a greedy heuristic algorithm to adapt the VNET overlay topology to the communication behavior of the application. VTTIF infers the application communication topology giving a traffic intensity matrix that is represented as an adjacency list where each entry describes communication between two VMs. The topology adaptation algorithm 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 this list by decreasing traffic intensity.
3. Establish the links in order until c links have been established.

The cost constraint c is supplied by the user or system administrator. The cost constraint can also be specified as a percentage of the total intensity reflected in the inferred traffic matrix, or as an absolute limit on bandwidth.[1]

Figure 8 illustrates topology adaptation. Here, an application configured with neighbor-exchange on a ring application topology of four VMs, starts executing with a VNET star topology (dotted lines) centered on the Proxy. VTTIF infers the topology and in response VADAPT tells VNET to add four links (dark lines) to form an overlay ring among the VNET daemons, thus matching the application's topology.

We refer to these added links as the *fast path topology*, as they lead to faster communication between the application components. It is important to note that

- The links may be of different types (TCP, UDP, STUN [33], HTTP, SOAP, etc) depending on the security policies of the two sites.
- Some links may be more costly than others (for example, those that support reservations).
- Not all desired links are possible.

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.

3.2 Migration

VADAPT uses a greedy heuristic algorithm to map virtual machines onto physical hosts. As above, VADAPT uses the application communication behavior as captured by VTTIF and expressed as an adjacency list as its input. In addition, we also use throughput estimates between each pair of VNET daemons arranged in decreasing order. The algorithm 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.

[1]The precise details of this algorithm (and the next) can be found on our website: http://virtuoso.cs.northwestern.edu/vadapt-algs-rev1.pdf.

3.3 Forwarding rules

Once VADAPT determines the overlay topology, we compute the forwarding rules using an all pairs shortest paths algorithm with each edge weight corresponding to the total load on the edge from paths we have determined. This spreads traffic out to improve network performance.

3.4 Combining algorithms

When we combine our algorithms, we first run the migration algorithm to map the VMs to VNET daemons. Next, we determine the overlay topology based on that mapping. Finally, we compute the forwarding rules.

4 Experiments with BSP

Our evaluation of VADAPT for bulk-synchronous parallel applications examines inference time, reaction time, and benefits of adaptation using topology adaptation, migration, and both. We find that the overheads of VADAPT are low and that the benefits of adaptation can be considerable. This is especially remarkable given that the system is completely automated, requiring no help from the application, OS, or developer.

4.1 Patterns

Patterns [14] 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.

4.2 Topology adaptation

In earlier work [40] we demonstrated that topology adaptation alone can increase the performance of patterns, although the evaluation was very limited. We summarize and expand on these results here. We studied all combinations of the following parameters:

- Number of VMs: 4 and 8.
- Application topology and communication patterns: neighbor exchange on a bus, ring, 2D mesh, and all-to-all.
- Environments: (a) All VMs on a single IBM e1350 cluster[2], (b) VMs equally divided between two adjacent IBM e1350 clusters connected by two firewalls and a 10 mbit Ethernet link, (c) VMs equally divided between one IBM e1350 cluster and a slower cluster[3] connected via two firewalls and a campus network, and (d) VMs spread over the wide area hosted on performance-diverse machines at CMU, Northwestern, U.Chicago, and on the DOT network[4].

Reaction time

For eight VNET daemons in a single cluster that is separated from the Proxy and user by a MAN, different fast path topologies and their default forwarding rules can be configured in 0.7 to 2.3 seconds. This configuration emphasizes the configuration costs. Creating the initial star takes about 0.9 seconds. Recall from Section 2.2 that the VTTIF inference time depends on the smoothing interval chosen and other parameters, with the best measured time being about one second. In the following, VTTIF is configured with a 60 second smoothing interval.

Benefits

If we add c of the n inferred links using the VADAPT topology adaptation algorithm, how much do we gain in terms of throughput, measured as iterations/second of patterns? We repeated this experiment for all of our configurations. In the following, we show representative results.

Figure 9 gives an example for the single cluster configuration, here running an 8 VM all-to-all communication. Using only the resilient star, the application has a throughput of $\sim$1.25 iterations/second, which increases to $\sim$1.5 iterations/second when the highest priority fast path link is added. This increase continues as we add links, improving throughput by up to factor of two.

Figure 10 illustrates the worst performance we measured, for a bus topology among machines spread over two clusters separated by a MAN. Even here, VADAPT did not decrease performance.

Figure 11 shows performance for 8 VMs, all-to-all, in the WAN scenario, with the hosts spread over the WAN (3 in a single cluster at Northwestern, 2 in another cluster at Northwestern, one in a third MAN network, one at U.Chicago, and one at CMU. The Proxy and the user are located on a separate network at Northwestern. Again, we see a significant performance improvement as more and more fast path links are added.

[2]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 switch.

[3]Nodes are dual 1 GHz P3s with 1 GB RAM running Red Hat 7.3 and VMware GSX Server 2.5, connected by a 100 mbit switch.

[4]www.dotresearch.org

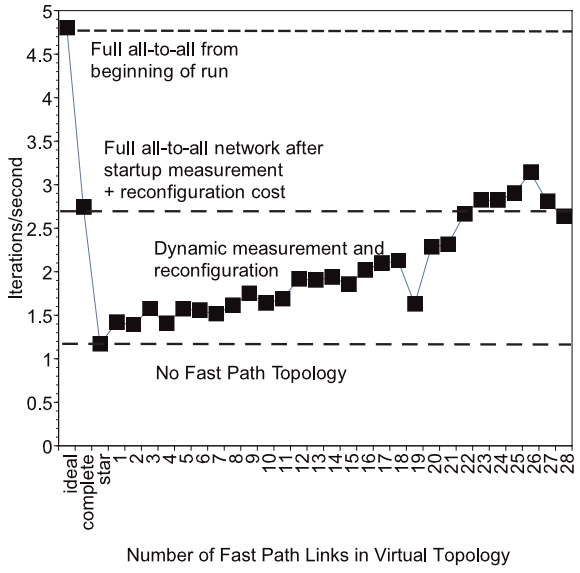

Figure 9. All-to-all topology with eight VMs, all on the same cluster.

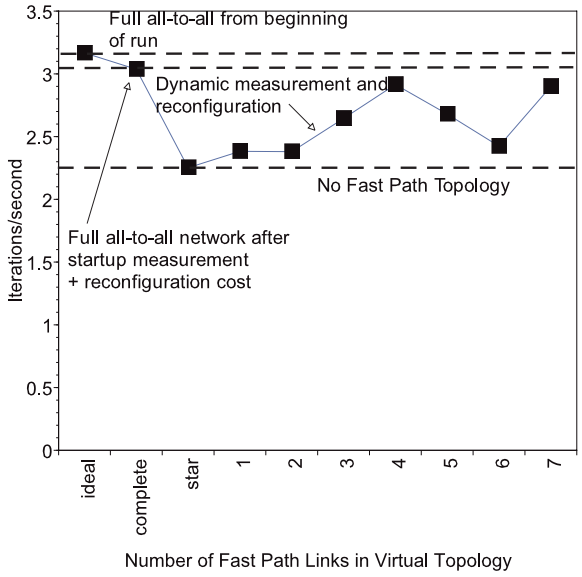

Figure 10. Bus topology with eight VMs, spread over two clusters over a MAN.

4.3 Migration and topology adaptation

Here we show, for the first time, results for migration and topology adaptation (Section 3), separately and together. 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.

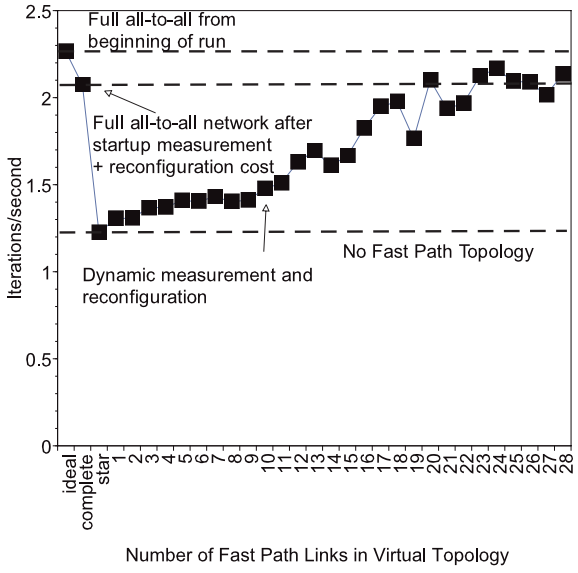

Figure 11. All-to-all topology with eight VMs, spread over a WAN.

Reaction time

The time needed by VNET to change the topology is as described earlier. The additional cost here is in VM migration. As we mentioned in the introduction, there is considerable work on VM migration. Some of this work has reported times as low as 5 seconds to migrate a full blown personal Windows VM [21]. Although Virtuoso supports plug-in migration schemes, of which we have implemented copy using SSH, synchronization using RSYNC [44], and migration by transferring redo logs in a versioning file system [5], in this work, we use RSYNC. The migration time is typically 300 seconds.

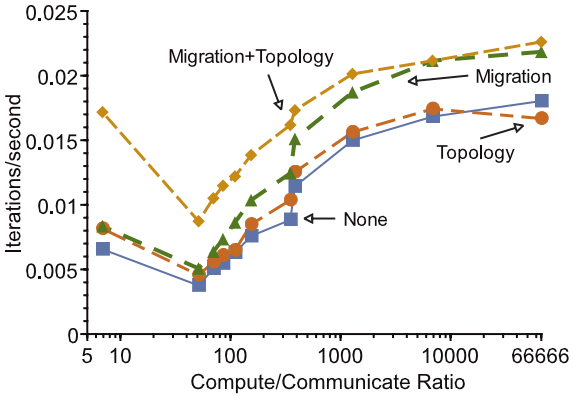

Figure 12. Effect on application throughput of adapting to compute/communicate ratio.

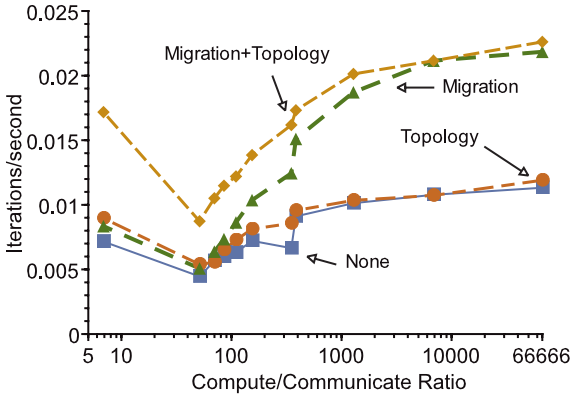

Figure 13. Effect on application throughput of adapting to external load imbalance.

Benefits

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 12 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. 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 13 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.

4.4 Scaling

We tested topology adaptation scenarios (Section 4.2) with all-to-all traffic among up to 28 VMs, the maximum possible on a single one of our clusters. 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 of VMs, thus the system will scale as VNET scales, not as migration scales. The number of forwarding rules per host can also grow with the square of the number of VMs, although the worst topology for this is the unlikely to be used linear topology. For an all-to-all topology, the number of forwarding rules per host grows linearly with the number of VMs. For the initial star topology, the total number of links and forwarding rules in the system grows linearly with the number of VMs.

With 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. 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 smaller scenarios we described previously.

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

4.5 Discussion

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

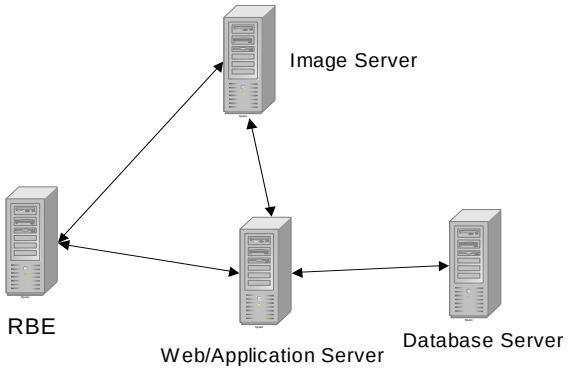

Figure 14. The configuration of TPC-W used in our experiment.

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 the level of abstraction also makes adaptation much more straightforward to accomplish.

Clearly it is possible to use our inference tool, VTTIF, the adaptation mechanisms of VNET, and the adaptation algorithms of VADAPT to greatly increase the performance of existing, unmodified BSP applications running in a VM environment like Virtuoso.

Adaptation needs to be sensitive to the nature of the application and different or multiple adaptation mechanisms may well be needed to increase performance. The inference capabilities of tools like VTTIF play a critical role in guiding adaptation so that maximum benefit can be derived for the application. While VTTIF tells us the application's resource demands, it does not (yet) tell us where the performance bottleneck is. This is an important next step for us. Determining the application's performance goal is also a key problem. In this work, we used throughput. More generally, we can use an objective function, either given by the programmer or learned from the user.

5 Multi-tier web sites

Can VADAPT help 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. We are still in the early stages of applying VADAPT to this domain, but we have promising results that indicate that considerable performance gains are possible.

TPC-W 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 14 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 backend 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 servers and the image server.

[5]We use the Wisconsin PHARM group's implementation [17], particularly the distribution created by Jan Kiefer.

	No Topology	Topology
No Migration	1.216	1.76
Migration	1.4	2.52

Figure 15. Web throughput (WIPS) with image server facing external load under different adaptation approaches.

The primary TPC-W metric is the WIPS rating. Figure 15 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.

6 Conclusions

We have demonstrated the power of adaptation at the level of a collection of virtual machines connected by a virtual network. Specifically, we can, at run-time, infer the communication topology of a BSP application or web site executing in a set of VMs. Using this information, we can dramatically increase application throughput by using heuristic algorithms to place the VMs on appropriate nodes and partially or completely match the application topology in our overlay topology. *Unlike previous work in adaptive systems and load balancing, 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.*

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] 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).

[3] 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).

[4] 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).

[5] 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).

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

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

[8] 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).

[9] FIGUEIREDO, R., DINDA, P., AND FORTES, J., Eds. *Special Issue On Resource Virtualization*. IEEE Computer. IEEE, May 2005.

[10] 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).

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

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

[13] 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.

[14] 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).

[15] GUPTA, A., ZANGRILLI, M., SUNDARARAJ, A., DINDA, P. A., AND LOWEKAMP, B. B. Free network measurement for adaptive virtualized distributed computing. In Submission.

[16] 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.

[17] 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).

[18] 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.

[19] 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).

[20] 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.

[21] 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.

[22] LANGE, J. R., SUNDARARAJ, A. I., AND DINDA, P. A. Automatic dynamic run-time optical network reservations. In *Proceedings of the 14th IEEE International Symposium on High-Performance Distributed Computing (HPDC)* (July 2005). In this volume.

[23] LIN, B., AND DINDA, P. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In Submission. A version of this paper is available as Technical Report NWU-CS-05-06, Department of Computer Science, Northwestern University.

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

[25] 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).

[26] 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.

[27] 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.

[28] LOWEKAMP, B., TIERNEY, B., COTTREL, L., HUGHES-JONES, R., KIELEMANN, T., AND SWANY, M. A hierarchy of network performance characteristics for grid applications and services. Tech. Rep. Recommendation GFD-R.023, Global Grid Forum, May 2004.

[29] 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.

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

[31] 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).

[32] 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* (December 2002).

[33] ROSENBERG, J., WEINBERGER, J., HUITEMA, C., AND MAHY, R. Stun: Simple traversal of user datagram protocol (udp) through network address translators (nats). Tech. Rep. RFC 3489, Internet Engineering Task Force, March 2003.

[34] 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* (December 2002).

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

[36] 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).

[37] 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.

[38] 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.

[39] 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.

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

[41] 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).

[42] 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.

[43] 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).

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

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

[46] 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.

[47] 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.

[48] WOLSKI, R., SPRING, N. T., AND HAYES, J. The network weather service: A distributed resource performance forecasting system. *Journal of Future Generation Computing Systems 15*, 5–6 (October 1999), 757–768.

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

[50] 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).

[51] 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.