Network Performance Analysis in Support of Virtual Clusters

Pacia Harper   Brian Fields

Computer Sciences Department
University of Wisconsin–Madison
{pacia, fields}@cs.wisc.edu

December 18, 2000

Abstract

Every business wants a web presence and are enlisting co-locating hosting services to provide it. Furthermore, e-businesses are growing to need the power and performance of cluster application and database software. The confluence of these two trends leads to the desirability of co-locating server farms that efficiently support cluster applications. We believe this need will be met by *virtual clusters*.

Virtual clusters give the illusion of ownership to each co-locating cluster application. Each application exists in its own environment and operating system. The virtualization is achieved by running virtual machines atop every physical machine and implementing policies via special virtual cluster management software.

In this paper, we analyze the network performance and contention through virtual machines, using VMWare [VMW00b] as the virtual machine software. The results collected give an indication of the performance degradation to be expected in a virtual cluster due to (a) the level of indirection through the virtual machines and (b) contention effects from different cluster applications sharing physical machines.

1 Introduction

These days web presence is critical or important for nearly every business: small, medium, or large. Most businesses do not have the people and equipment resources to administer and maintain their own web servers. Instead, they contract service providers to provide this service. The service providers store racks of machines in large server farms and use software, such as Ensim's ServerXchange [Ens00], to co-locate applications for the different businesses.

Meanwhile, cluster packages—such as the Oracle9i suite [Ora00]—are targeting e-businesses with middle-tier and database services. As more businesses grow to need this degree of power and performance, it will be increasingly important for co-locating hosting service providers to efficiently support cluster applications. Specifically, multiple cluster applications from multiple clients should be able to seamlessly execute on the same set of machines, efficiently and fairly utilizing the available computational power.

One solution is to give each cluster application the illusion of having its own cluster, even if it is sharing machines with other applications. This solution allows a single physical cluster to run cluster applications requiring different environment configurations, or even different operating systems entirely. This can be accomplished through *virtualization* of the hardware, as has been done for user applications on a single workstation [VMW00b]. A *virtual cluster* (VC) would virtualize each individual machine and use management software to intelligently assign processes of a cluster application to particular virtual machines.

Many issues arise in the design of virtual cluster software. Chief among them is maintaining good performance for individual cluster applications despite the contention effects of multiplexing resources between them and the extra level of indirection due to the virtual machine software. These effects can, for instance, degrade computation performance, disk performance, or network performance.

In this paper, we analyze network performance. Good network performance, especially latency, is crucial for getting good performance from cluster applications. We use the VMWare virtual machine [VMW00b] to better understand characteristics critical to virtual cluster software. We start by describing the VMWare network architecture and the potential sources of contention (Section 3). We then quantify the overhead of accessing the network through a virtual machine as well as contention effects due to the shared driver and network interface (Section 4). Finally, we propose a solution to ensure performance isolation between virtual machines (Section 5) and conclude in Section 6.

2 Related Work

The related work is divided into two categories: previous work with virtual machines and performance isolation. We survey each in turn.

2.1 Virtual Machines

Virtual machines have a history going back to the 60's and 70's [PG74, CFH$^+$80] with several well-publicized commercial products from IBM [Ano99]. Recently, they have become very popular with the advent of VMWare [VMW00b]. This software allows users to run multiple OSes,*e.g.*Windows and Linux, simultaneously on one machine. We used VMWare as the virtual machine for our experiments and detail its network architecture in the next section.

Virtual machines have also been used recently to solve difficult problems in large-scale shared memory multiprocessors. DISCO [BDGR97, GTHR00] bypasses the requirement of a scalable OS (capable of dealing with NUMA issues) by enabling multiple operating systems to co-exist simultaneously on a large-scale NUMA multiprocessor. They also showed how a virtual machine that induced very little overhead could be implemented atop a modern processor. Virtual clusters are similar to this except that the substrate is different: a cluster of workstations as opposed to a NUMA multiprocessor. In addition, while DISCO tries to make a NUMA seem like an SMP, VCs simply try to make a shared cluster look like an owned cluster.

2.2 Performance Isolation

Banga et al. [BDM99] described the current weakness of performance guarantees in OSs such as Unix. The combination of priorities and timeslices in Unix allows some control over CPU sharing, but work done in the kernel is not accounted to the customer (application, web client, etc.) requesting the work. They propose an OS abstraction called *resource containers* to associate all the work done on behalf of a given customer with one account (container). When a customer consumes more than its fair share of resources, the customer's priority decays. Resource containers could be used for a variety of resource accounting, but the authors implemented it only for CPU time. One advantage of resource containers is that they separate the logical unit of resource allocation from the process. With virtual clusters, the logical unit of resource allocation is the virtual machine. However, problems of accounting for resource usage in the virtualization layer remain.

Verghese et al. [VGR98] show how SMP operating systems do not have sufficient resource management for multi-user multiprocessors, permitting one user to affect another user in an unpredictable way. They then describe performance isolation schemes that remedy this problem for three resources: the CPU, memory, and disk bandwidth. Although they do not discuss isolation for network bandwidth, many of the principles applied to disk bandwidth apply to the network bandwidth as well. This is true because, for both, measuring the instantaneous bandwidth is not possible. They employ a decaying function that observes the bandwidth consumed over some short interval in the recent past and devise a *fairness* criteria based on that. Enforcing fair sharing is easier for network I/O than for disk I/O, since disks have the added complication that a policy of strict fairness may hurt performance by increasing seek times.

U-Net [vEBBV95] used user-level network interfaces to improve network latency by bypassing the kernel calls and software-stack traversal required in normal networking. The U-Net implementation gives each application (depending on the number of applications) an endpoint on the network interface. Assuming the endpoints are serviced round-robin, this provides a degree of network performance isolation. (Applications could possibly starve their peers by sending very large messages, depending on the MTU of the underlying network.) However, network endpoints are a limited resource. This can lead to several applications being multiplexed onto one, kernel-owned endpoint. In the case of virtual clusters, the applications are virtual machines, and it is possible for several VMs on one machine to be part of the same virtual cluster. In this case, we might want finer control of network resource allocation (*e.g.*to share the NIC evenly between VCs rather than VMs).

InfiniBand [Inf00] is a new system-area networking standard, greatly influenced by U-Net [vEBBV95] and later user-level networking ideas, such as the Virtual Interface Architecture [DRM$^+$98]. Like U-Net and VIA, InfiniBand gives a finite number of applications direct access to the network interface. The comments on U-Net apply to InfiniBand as well.

However, InfiniBand gives software much greater control over the network. For example, each cluster has a subnet manager, and this manager is expected to set various quality-of-service parameters in the network switches themselves. This introduces the possibility of controlling network resource utilization in a global way, rather than only at the endpoints. However, it also introduces new challenges for virtualization. If a virtual cluster is running on virtual InfiniBand, the cluster will "expect" network latencies and behavior to correspond with the configuration of the virtual network and switches.

3 VMWare Network Architecture

For our network performance analysis we used VMWare Workstation virtualization software [VMW00b]. The basic network architecture of VMWare is shown in Figure 1. All the device drivers, in the host OS and guest OSes, are standard Ethernet drivers.

VMware includes a kernel module, VMNet, which passes packets between the host OS and the device drivers on the guest OSes. VMNet intercepts packets sent from the guest OSes and forwards them on to the host's device driver. Incoming packets are sent to VMNet as if to a normal kernel software stack, such as TCP. VMNet then demultiplexes the packets to the appropriate guest OS. There are two potential communication bottlenecks in this diagram: first, the VMNet module; second, the device driver and Ethernet card on the physical host.

4 Experiments

For our experiments, we used two x86 machines running Linux. We installed VMware on each machine, and created two virtual machines on each machine. On each machine, we could have no VMs running, one running, or both running. We set the VMs up with bridged networking, in which the VMNet module behaves like a bridge. In this configuration, the VMs appear to the outside world like real machines—they received ARP messages, and we could telnet to them from the workstations on our desks.

We ran two types of experiments. First, we attempted to characterize the basic VMware network performance. Second, we attempted to create network contention between virtual machines. We hypothesized that multiple virtual machines on a shared phyiscal machine would contend for

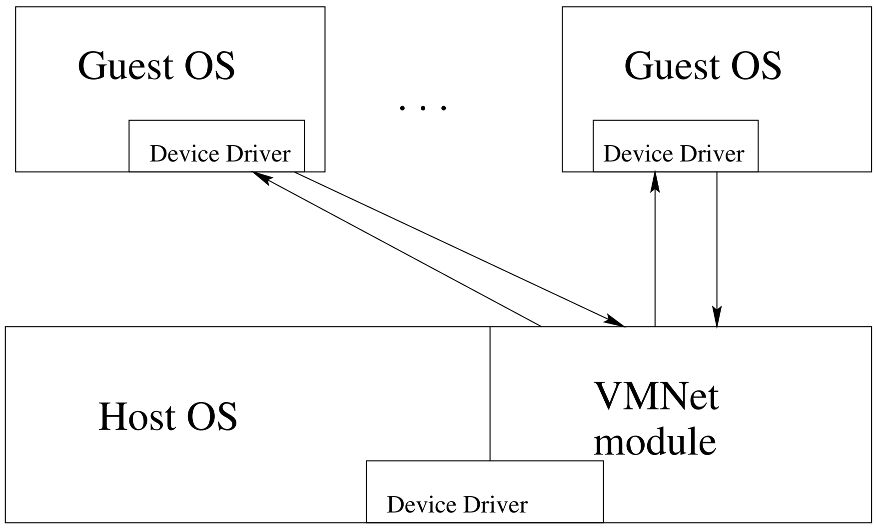


Figure 1: **VMWare Network Architecture.** Potential bottlenecks include the VMNet module and the interface to the Ethernet card.

network resources.

4.1 Software Used

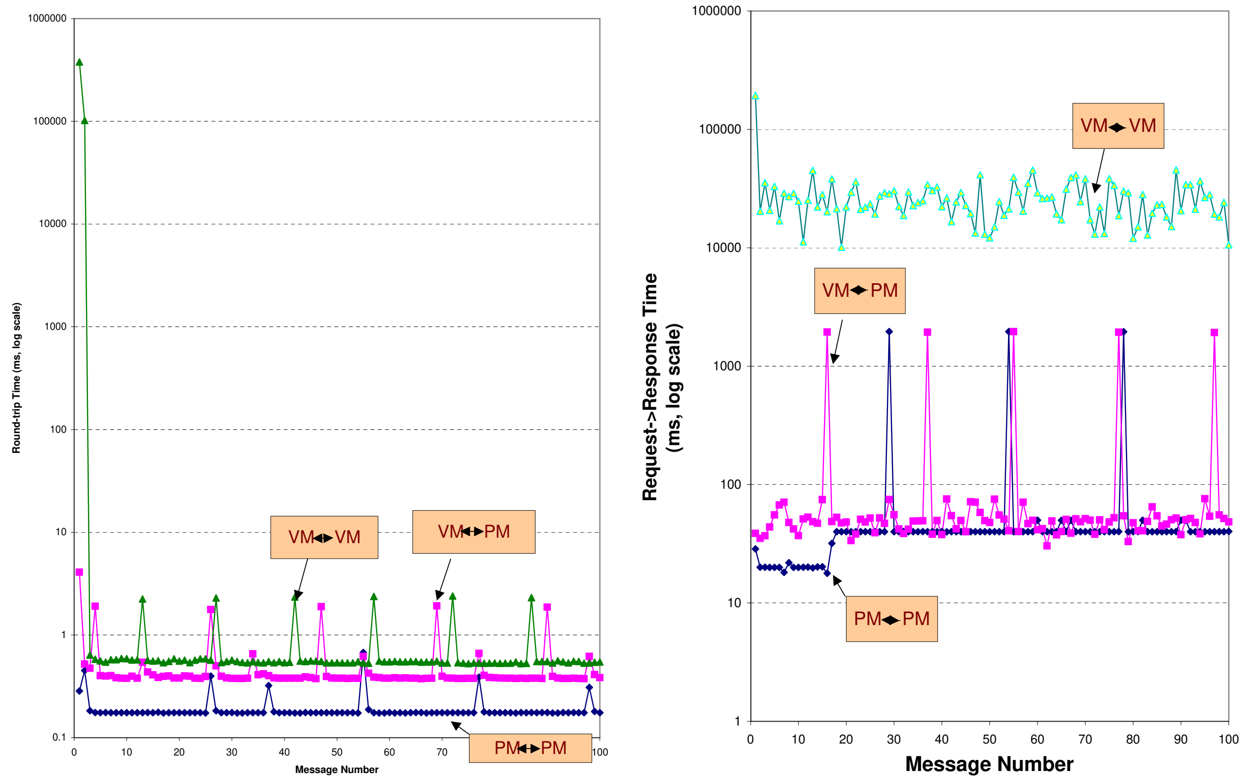
In both sets of experiments, we used a simple client-server program. The client and server communicate using TCP/IP. The client and server each take the arguments:

```
<message size in bytes> <number of messages>
```

When the client starts, it connects to the server. (The server is started first, so it can listen on the server port.) After the connection is established, the client sends one message of size `<message size in bytes>`, then waits for the server's response. The client repeats this until it has sent `<number of messages>`.

The client prints the send, wait, receive, and request-response (RTT) times for each message, as well as printing the overall mean for each time category at the end. The server also provides these statistics (aside from request-response times). Since the client and server are symmetrical, we present only the client results.

In all cases presented here, we set `<number of messages>` to 100, and varied only the message size. In all cases, both the client and the server were given the same message size. This is of course not representative of genuine network traffic. We had hoped to use the Surge web traffic generator to do more realistic experiments. However, the performance results we did get tended to indicate that running a heavy-duty, high-traffic benchmark on a VM would be slow and uninformative.



(a) With small (100 byte) messages.

(b) With large (100,000 byte) messages.

Figure 2: **Round-trip Times.**

We would like to note here that our results improved dramatically on December 17th, just before the deadline of this paper. Despite a great deal of experiment since then, we still do not understand why the results changed. Although the results we present here are genuine, they may be misleading as to VMware's true network performance.

4.2 Basic Performance

For the experiments in Figure 2, we ran only one VM on each physical machine. We considered three cases. In all three, the client and server are on separate physical machines (*i.e.*, all traffic goes over the real Ethernet).

1. Both the client and the server are running on virtual machines.
2. The client is running on a virtual machine, and the server is running on a physical machine.
3. Both the client and the server are running on physical machines (no virtualization).

At first, we ran messages of 100 bytes each. Later on, we began running 100,000-byte messages, in order to explore the network performance more. When the 100,000-byte message performance turned out to be much worse in case 1 than in case 2, we were puzzled—it seemed that case 1 should take at most twice as long as case 2. In case 2 the message must traverse the VMNet module just once, whereas in case 1 it must traverse VMNet twice.

We wanted to understand the peculiar behavior of the large messages better, so we did several things. First, we also began running tests with messages of 25,000 bytes, 50,000 bytes, and 75,000 bytes. This made the situation even more perplexing. Although the environment and software was identical, exchanging 200 50,000-byte messages took an order of magnitude longer than exchanging 200 100,000-byte messages. The average RTT for 25,000 bytes was about 25% of the average RTT for 100,000 bytes, which made sense. However, it was as if the 50,000 and 75,000 byte cases had been run on completely different (and very slow) machines.

Second, we modified the VMNet module slightly, to give us a look at the traffic going between the client and the server. (Although VMware is a closed-source commercial product, source code for the kernel modules is provided.) This modification was in three stages. First, we developed a method for detecting client-server packets as they passed through the module functions. We did this by choosing unique default ports for the client and server. Since the client and server communicate over TCP/IP, the client and server ports appear in a predictable place in the header of each message. Detecting these packets allowed us to ignore the (surprisingly) large number of other messages processed through VMNet.

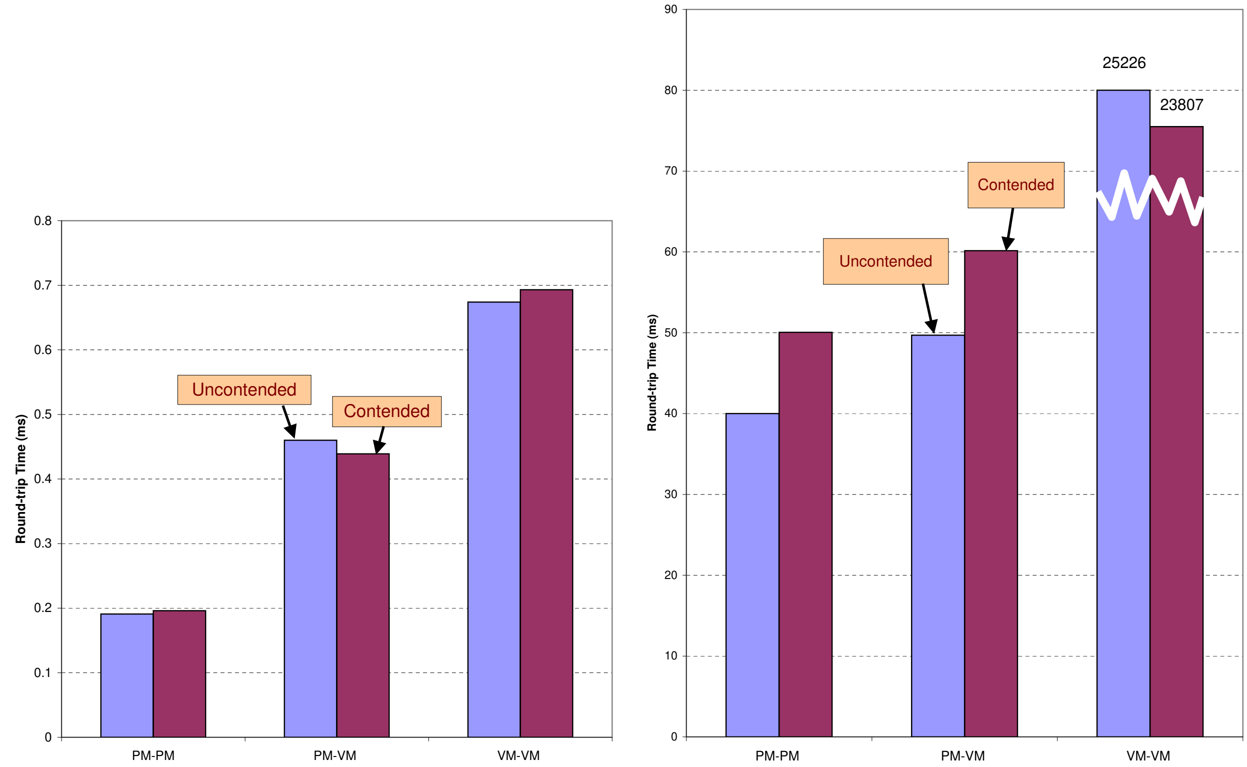
Second, we modified each VMNet function that processed packets to time its own execution. We were looking to see if some function(s) took an inordinately long time. This experiment was a success in that we got the desired output. However, it was a failure in that all of the functions took well under a millisecond.

Finally, we modified VMNet to count the bytes in packets sent by the client. This was in order to test the hypothesis that the TCP stack on the guest OS was misbehaving—perhaps doing excessive timeouts and retransmissions. The results were peculiar, but did not allow us to fully account for the poor performance: while the PM-PM experiments transferred data precisely corresponding to the specified size of the packets, the VM-VM runs transferred approximately twice that amount. This suggests some re-transmissions during the VM-VM runs, but not enough to account for the large performance disparity. This may be a symptom of the larger problem, however.

4.3 Contention Results

In the experiment for Figure 3, we ran two client-server pairs simultaneously, in an attempt to create contention. Again, we tested this with three cases, which are analogous to the three cases used above. We ran one client-server pair indefinitely, and the other pair for 100 messages. We measured the pair that executed 100 messages; the other pair was used solely to create network contention.

In all experiments, all four processes (client 1, client 2, server 1, server 2) were sending the same message size. We ran the experiments with 100-byte messages and then with 100,000-byte



(a) With small (100 byte) messages. **(b)** With large (100,000 byte) messages.

Figure 3: **Contention between virtual machines, physical machines, and across virtual-to-physical machines.**

process	machine
client 1	crash1vm1
client 2	crash1vm2
server 1	crash2vm1
server 2	crash2vm2

Figure 4: Case 1 for the contention tests.

messages.

1. Both clients are running on the same physical machine, but different virtual machines. Both servers are running on the same physical machine (different from the clients), but different virtual machines. This is displayed in figure 4.
2. Like 1, but both servers are running directly on a physical machine (*i.e.*, crash2).
3. Both clients and both servers are running directly on physical machines. The clients are still on one machine (*i.e.*, crash1) and the servers on another (*i.e.*, crash2).

The results here were somewhat disappointing. Although cases 2 and 3 (VM$<->$physical and physical$<->$physical) showed up to a 25% performance hit from contention, the VM$<->$VM case showed no effect. We speculate that the absolute performance hit from contention is about the same in all three cases—about 10 ms for 100,000-byte messages. However, the relative difference is much smaller for the VM$<->$VM case, because the average RTTs for that case are so large.

5 Solutions

5.1 Providing Performance Isolation

Recently, VMWare announced a way to provide some network performance isolation by replication of the VMNet module (one for each virtual machine) [VMW00a]. This technique eliminates one possible site of contention, the VMNet interface. However, each VM must still go through the same device driver and network interface (assuming traditional, non-user-level networking). This puts VMs on the same footing as applications running on the physical machine. It improves matters in that VM communications no longer go through two potential bottlenecks (VMNet and the NIC). However, VMs now get the same performance isolation guarantees as applications running on the host OS—which to our knowledge means no guarantees at all.

Whether normal host OSs should provide network performance isolation to their applications is an open question. Given the availability of high-performance user-level-networking interfaces, the question is probably also moot; users running very demanding network applications will use ServerNet, InfiniBand, Quadrics, Myrinet, ...

However, high-performance hardware does not fix the problem for virtual machines, and so providing network performance isolation in the virtualization layer makes sense. High-performance virtualization layers will run directly on hardware, without a host OS, so in this case the issue of host OS guarantees is irrelevant anyway.

So how can we provide performance isolation in the virtualization layer? The insight here is that the VMNet module is essentially a private, virtual network. (Not, of course, a virtual private network.) This means that various techniques used for flow control in real networks—such as virtual channels [Dal92] and Random Early Detection [FJ93]—can be used.

For this purpose, a "flow" would be a stream of messages from one VM, possibly to many different sources. This would introduce the complication of identifying flows correctly—the simplest method would be for the virtualization layer to tag each message appropriately before passing it on to the VMNet module.

However, VMNet (or a similar virtual network used by the virtualization layer) can access much more information than is available to switches or routers (even those that support virtual channels). A more ambitious approach would take advantage of the ability to deliver extra information to VMNet. There is still more potential in the possibility of modifying the guest OS itself, or perhaps only the guest's device driver. The current VMNet provides communication between VMNet and the guest OS device drivers, but only one party (VMNet) is aware of the real situation. If both parties knew what was going on, they could collaborate in ways that are impossible now.

5.2 Determining Sharing Policies

An issue frequently raised in systems research is what policies need to be enforced, and which part of the system sets them. The most obvious way of providing network performance isolation to virtual clusters is to share network resources equally between all VMs on a given physical machines. However, this raises several questions.

First, is it best to control sharing only at the endpoints? That is what we have discussed to this point, and with current mainstream hardware it is difficult to control sharing at any other level. The new Infiniband [Inf00] standard has features that would allow sharing to be controlled and tuned throughout the network, not just at the endpoints.

Second, what is it that we are sharing? Is it a matter of VMs that share a physical machine splitting the local network resources between them? Or do we need to take a global perspective, sharing the network resources of the entire cluster between a set of VCs? It is not obvious how local behavior will affect global behavior, and vice versa. Trying to control sharing at a global level implies some kind of centralized entity, which hurts scalability. On the other hand, locally optimal decisions may not produce a globally optimal outcome.

Finally, how flexible should the sharing policies be? Enforcing sharing at the virtual-machine level is much easier than enforcing it at the virtual-cluster level. However, this means that a VC with 12 nodes could use twice as much of the network resources as a VC with 6 nodes. That may or may not be what we want. Should VCs or VMs have a fixed allocation of network resources throughout their lifetimes, or should they gain more resources as they become available? The first option makes sense, as clusters usually have fixed network resources. However, the second option could lead to better performance, at the risk of destabilizing resource-aware protocols such as TCP.

6 Conclusion and Future Work

Our analysis of network performance through virtual machine software has revealed a great deal of unpredictable behavior. In particular, transmitting large messages between virtual machines resulted in extraordinarily poor performance during most of our experiments. Inexplicably, the performance improved just before the paper submission deadline. This suggests more analysis is needed to determine what is causing erratic behavior within VMWare (or if it was due to some interaction with our experimental setup), so as to avoid such anomalies in future versions of virtual machine software. This strange behavior is not likely due to a bug in our experimental software as the poor performance was observed with a simple *ping* early on in the project.

Immediate future work would include measuring VMWare's performance as it is being observed now: looking at the overhead, contention, etc. The next step would include running real cluster software to see how the traffic patterns influence the results. From this data, it should be more clear whether an advanced solution to network performance isolation, such as that presented in Section 5 is worthwhile to implement.

References

[Ano99] Anonymous. Short history of IBM's virtual machines. 1999.

[BDGR97] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447, November 1997.

[BDM99] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd Symposium on Operating Systans Design and Implementation (OSDI-99)*, pages 45–58, Berkeley, CA, February 22–25 1999. Usenix Association.

[CFH+80] M. D. Canon, D. H. Fritz, J. H. Howard, T. D. Howell, M. F. Mitoma, and J. Rodriguez-Rosell. A virtual machine emulator for performance evaluation. *Communications of the ACM*, 23(2):71–80, February 1980.

[Dal92] W. J. Dally. Virtual-channel flow control. *IEEE Trans. on Parallel and Distributed Systems*, May 1992.

[DRM+98] Dave Dunning, Greg Regnier, Gary McAlpine, Don Cameron, Bill Shubert, Frank Berry, Anne Marie Merritt, Ed Gronke, and Chris Dodd. The virtual interface architecture. *IEEE Micro*, March/April 1998.

[Ens00] Inc. Ensim. Building a profitable hosting business with Ensim ServerXchange. *White Paper [http://www.ensim.com/pdf/wp.pdf]*, 2000.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[GTHR00] Kingshuk Govil, Dan Teodosiu, Yongqiang Huang, and Mendel Rosenblum. Cellular disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 18(3):229–262, 2000.

[Inf00] InfiniBand Trade Association. Infiniband specification 1.0. *[http://www.infinibandta.org]*, 2000.

[Ora00] Oracle. Oracle9i real application clusters. *[http://www.oracle.com/ip/deploy/database/9i/continuity/i* 2000.

[PG74] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, July 1974.

[vEBBV95] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-net: A user-level network interface for parallel and distributed computing. In *Proceedings of the 15th Symposium on Operating Systems Principles (15th SOSP'95), Operating Systems Review*, pages 40–53, Copper Mountain, CO, December 1995. ACM SIGOPS. Published as Proceedings of the 15th Symposium on Operating Systems Principles (15th SOSP'95), Operating Systems Review, volume 29, number 5.

[VGR98] Ben Verghese, Anoop Gupta, and Mendel Rosenblum. Performance isolation: Sharing and isolation in shared-memory multiprocessors. *ACM SIGPLAN Notices*, 33(11):181–192, November 1998.

[VMW00a] VMWare Inc. VMWare Newsletter. *Issue VI*, December 2000.

[VMW00b] VMWare Inc. VMWare virtual machine technology. *White Paper [http://www.vmware.com/pdf/whitepaper.pdf]*, September 2000.