Fast Transparent Migration for Virtual Machines

Michael Nelson, Beng-Hong Lim, and Greg Hutchins

VMware, Inc.
Palo Alto, CA 94304

Abstract

This paper describes the design and implementation of a system that uses virtual machine technology [1] to provide fast, transparent application migration. This is the first system that can migrate unmodified applications on unmodified mainstream Intel x86-based operating system, including Microsoft Windows, Linux, Novell NetWare and others. Neither the application nor any clients communicating with the application can tell that the application has been migrated. Experimental measurements show that for a variety of workloads, application downtime caused by migration is less than a second.

1 Introduction

Fast transparent migration can improve global system utilization by load balancing across physical machines, and can improve system serviceability and availability by moving applications off machines that need servicing or upgrades. This paper describes a migration system, named *VMotion*, that has been shipping since 2003 as an integral part of the VMware VirtualCenter product [2]. Future VMware products will utilize *VMotion* to automate load balancing across large numbers of servers.

This paper makes the following contributions.

- It describes the first system to provide transparent virtual machine migration of existing applications and operating systems; neither the applications nor the operating systems need to be modified.
- It is the first paper to provide performance measurements of hundreds of virtual machine migrations of concurrently running virtual machines with standard industry benchmarks.
- It characterizes the overheads and resources required during virtual machine migration.

2 Virtual Machine Migration

Virtual machine migration takes a running virtual machine and moves it from one physical machine to another. This migration must be transparent to the guest operating system, applications running on the operating system, and remote clients of the virtual machine. It should appear to all parties involved that the virtual machine did not change its location. The only perceived change should be a brief slowdown during the migration and a possible improvement in performance after the migration because the VM was moved to a machine with more available resources.

The migration system presented in this paper is part of the VMware VirtualCenter product that manages VMware ESX Server [3]. VMware ESX Server consists of two main components that implement the virtual platform: the virtual machine monitor (VMM) and the vmkernel. A guest operating system such as Windows or Linux runs on top of this virtual platform (see Figure 1). The VMM handles the execution of all instructions on the virtual CPU and the emulation of all virtual devices. The vmkernel schedules the VMM for each virtual machine and allocates and manages the resources needed by the virtual machines.

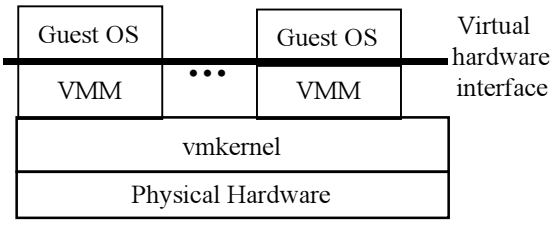

Figure 1. VM platform layers in VMware ESX Server.

Virtual machines provide a natural platform for migration by encapsulating all of the state of the hardware and software running within the virtual machine. There are three kinds of state that need to be dealt with when migrating a VM:

1) The virtual device state including the state of the CPU, the motherboard, networking and storage adapters, floppy disks, and graphics adapters.
2) External connections with devices including networking, USB devices, SCSI storage devices, and removable media such as CD-ROMs.
3) The VM's physical memory.

The actual migration process involves several steps:

1) Initiating the migration by selecting the VM to migrate and its destination.
2) Pre-copying the memory state of the VM to the destination while the VM is running on the source.

3) Quiescing the VM and sending the non-memory state.
4) Transferring control of the VM to the destination and resuming it at the destination.
5) Sending any remaining memory state and removing the dependency on the source machine.

The remainder of this section describes the steps involved in migrating three of the most important components of a VM: networking, SCSI storage devices, and physical memory.

Networking In order for a migration to be transparent all network connections that were open before a migration must remain open after the migration completes. The VMware ESX Server virtual networking architecture makes this possible.

A virtual Ethernet network interface card (VNIC) is provided as part of the virtual platform. Like a physical NIC, the VNIC has a MAC address that uniquely identifies it on the local network. Each VNIC is associated with one or more physical NICs that are managed by the vmkernel. The VNICs of many VMs can be attached to the same physical NIC.

Since each VNIC has its own MAC address that is independent of the physical NIC's MAC address, virtual machines can be moved while they are running between machines and still keep network connections alive as long as the new machine is attached to the same subnet as the original machine.

SCSI Storage We rely on storage area networks (SAN) or NAS to allow us to migrate connections to SCSI devices. We assume that all physical machines involved in a migration are attached to the same SAN or NAS server. This allows us to migrate a SCSI disk by reconnecting to the disk on the destination machine.

Physical Memory The physical memory of the virtual machine is the largest piece of state that needs to be migrated. Pausing a VM while the entire memory state is transferred will result in the VM being inaccessible for too long. We address this problem by copying the physical memory state from the source machine to the destination machine while the VM is running. This is possible because of the way that we manage the physical memory of the VM [3].

Each virtual machine expects to have a fixed set of physical address ranges that map to physical memory. VMware ESX Server dynamically allocates the real machine's physical memory to the running virtual machines. This requires adding a level of indirection to provide the physical memory layout expected by a guest operating system. All direct accesses to the VM's physical memory, as well as all writes to memory mapping hardware and page tables, are intercepted by the VMM. The VMM then translates these physical addresses into the actual machine addresses. Once the VM's memory mapping hardware and page tables are properly set up, the VM can run without any additional physical-to-machine address translation overhead.

We use this level of indirection to iteratively pre-copy the memory [4] while the VM continues to run on the source machine. The first step copies over the entire physical memory of the VM. Before each page is copied, it is marked read-only so that any modifications to the page can be detected by the VMM. When the first step is completed, some memory will have been modified by the still-running VM. The modified pages are then copied again to the destination while the VM continues to run. This procedure is then repeated until either the number of modified pages is small enough or there is insufficient forward progress. Currently we terminate the pre-copy when there are less than 16 megabytes of modified pages left or there is a reduction in changed pages of less than 1 megabyte.

3 Performance Measurements

This section investigates the performance characteristics of the virtual machine migration scheme described above. It presents measurements of the time to migrate a VM and the period during which the VM is unavailable. It also characterizes the effect of CPU resource allocation. Most importantly, it shows that for a variety of workloads VM migration can be fast and transparent to applications and operating systems.

3.1 Experimental Setup

All experiments were performed on a pair of identical Dell 1600SC servers each with two 2.4GHz Intel Xeon processors and 1 GB of RAM. The servers are connected to an EMC CLARiiON SAN via Qlogic 2300 HBAs. Intel Pro/1000 Gbit NICs are used to transfer the state of the VMs.

Each experiment migrates a single VM 50 times between two servers with 5-second intervals. The numbers reported are the average of the 50 migrations.

The experiments use the following VM workloads:

- ***idle***: An idle Windows 2000 Server.
- ***kernel-compile***: Linux kernel compilation in RedHat 7.2.
- ***iometer***: Iometer [7] running on Windows 2000 Server. Iometer was configured to run disk I/O

with 3 worker threads each performing I/O to a 500MB file with up to five outstanding writes.

- ***memtest86***: Memtest86 [5], which continuously reads and writes memory, running test #1 in a loop
- ***dbhammer***: Database Hammer [6], a client/server database load generator running on Windows 2000 Server. The server was migrated while the client was running on a third physical machine.

Except for the measurements in Section 3.4, the VM being migrated was the only VM running on the source machine and there were no VMs running on the destination machine.

3.2 Migration Time

Migration proceeds in several distinct steps. We are most interested in the downtime during which the VM is unavailable. This period must be short enough to avoid any noticeable loss of service from the VM. We are also interested in the total end-to-end time of a migration during which machine resources are consumed to perform the migration.

Downtime The total downtime consists of the time necessary to quiesce the VM on the source, transfer the device state to the destination, load the device state, and copy over all the remaining memory pages concurrently with loading the device state.

Figure 2 shows that the total downtime is less than one second for all workloads except *memtest86* and rises minimally with increasing memory sizes. *memtest86* is a pathological case where all the memory was modified during the pre-copy so that the VM downtime equals the time necessary to send the VM's entire memory.

End-to-end Time Figure 3 shows that the total end-to-end time depends strongly on the size of the VM's

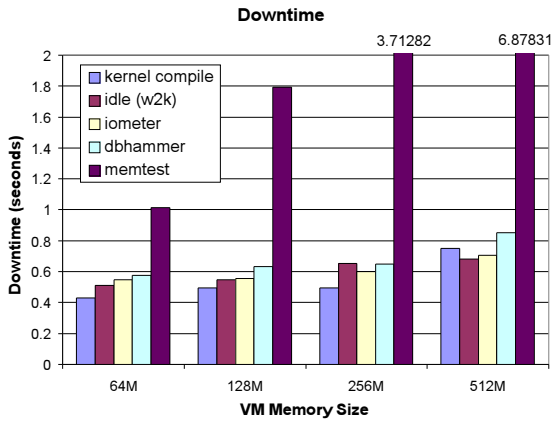

Figure 2: Downtime during migration for various workloads and VM memory sizes.

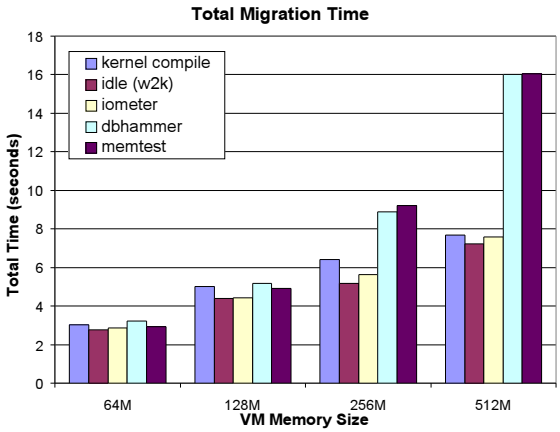

Figure 3. Total end-to-end time for a migration.

memory, and confirms the need to keep the VM running during most of this time. With pre-copying, the VM continues to run while memory is being transferred to the destination.

The number of pre-copy iterations required to migrate each workload was small. All workloads except for *memtest86* took 1 or 2 rounds before the number of modified pages was small enough to terminate the pre-copy. It took 2 or 3 rounds before the pre-copy was aborted because of lack of progress for *memtest86*.

3.3 Effect of Pre-copy

Figure 4 shows the effect of pre-copying memory on network throughput as measured by the *dbhammer* client during a window of three back-to-back migrations of the *dbhammer* server. The three large drops in throughput correspond to the downtime.

The smaller 20% drops in throughput are caused by the

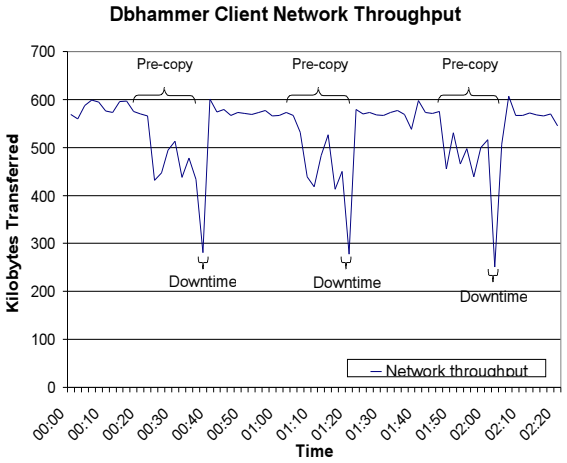

Figure 4. Effect on network throughput as seen by the client of a migrating *dbhammer* server.

pre-copy. This drop is caused by the overhead of marking the pre-copied pages as read-only, which involves halting all virtual CPUs, and the overhead of handling any protection faults to the read-only pages.

3.4 Resource Management

To monitor the effect of resource management, the source physical machine was loaded with the equivalent of 20 CPU-bound virtual machines, and the time to migrate an idle 512MB Windows 2000 Server VM was measured under different resource reservations. Figure 5 shows that reserving 30% of a CPU for migration minimizes the pre-copy time. This implies that it takes around 30% of a CPU to attain the maximum network throughput over the gigabit link.

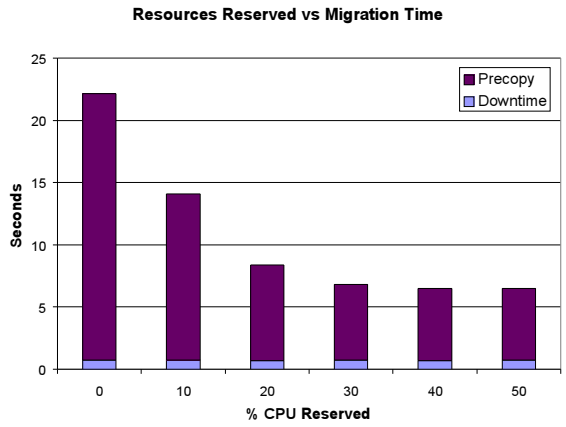

Figure 5. Effect of CPU reservation on migration from a heavily loaded source machine.

Even though the pre-copy time increased when insufficient CPU was reserved for the migration, the downtime remains small regardless of the amount of reserved CPU. It requires little CPU time to quiesce the VM and transfer over the non-memory state.

4 Related Work

There is large amount of previous work done on transparently migrating processes. Zap [8] is a recent system that provides process migration and contains a good discussion of previous work in this area.

The only other system that migrates virtual machines is described by Hansen and Jul in [9]. The fundamental difference between their work and ours is that they require guest OS modifications in order to perform the migration. Whereas our system can migrate any OS that runs on the Intel x86 architecture including closed-source operating systems such as Microsoft Windows, their system can only migrate operating systems that can be modified to work in their environment.

5 Conclusions

Previous attempts at application migration have had limited success primarily because of the difficulty of encapsulating the state of a running application. Virtual machines solve this problem by allowing not only an application to be encapsulated, but the operating system and the hardware as well. We have described a migration implementation that allows an entire running VM to be migrated from one physical machine to another. The migration is completely transparent to the application, the operating system and remote clients.

The method for migrating the physical memory of a VM is critical to providing transparent migration. It takes many seconds, even over fast networks, and significant CPU resources to transfer large memories. We have shown that transferring the memory while a VM is running minimizes the downtime. Our measurements show that a VM normally experiences less than one second of down time. Also, the end-to-end time of the migration and the impact on other VMs running on the machines involved in the migration can be controlled by properly managing CPU resources.

References

1. R. Goldberg. "Survey of Virtual Machine Research," *IEEE Computer*, 7(6), June 1974.
2. "Building Virtual Infrastructure with VMware Virtual Center," http://www.vmware.com/pdf/vi_wp.pdf.
3. C. Waldspurger. "Memory Resource Management in VMware ESX Server," *Proc. Of the 5th Operating Systems Design and Implementation*, December 2002.
4. M. Theimer, K. Lantz, and D. Cheriton. "Preemptable Remote Execution Facilities for the V-System," *Proc. Of the 10th Symposium on Operating System Principles*, December 1985.
5. "Memtest86 - A Stand-alone Memory Diagnostic," http://www.memtest86.com/.
6. "Microsoft SQL Server: Resource Kit," http://www.microsoft.com/sql/techinfo/reskit.
7. "Iometer Project," http://www.iometer.org/.
8. S. Osman, et al. "The Design and Implementation of Zap: A System for Migrating Computing Environments," *Proc. Of the 5th Operating Systems Design and Implementation*, December 2002.
9. J.G. Hansen and E. Jul, "Self-migration of Operating Systems," *Proc. Of the 11th ACM European SIGOPS Workshop,* September 2004.