ABSTRACT
Multiprocessor virtual machines (VMs) allow guest operating systems to use symmetric multiprocessing (SMP) in a VM. However, the use of SMP in a VM complicates CPU scheduling by the virtual machine monitor and can significantly increase the performance overhead of virtualization. This paper analyzes the performance of SMP virtualization used in two leading virtualization systems: VMware ESX 3.5 and Xen 3.2. Each is analyzed using single-threaded (SPEC CPU2006 445.gobmk) and multithreaded (SPEC OMP2001 332.ammp) CPU bound workloads to measure how the virtualization overhead scales when running SMP VMs. ESX has more overhead, but performs equally well for both single-threaded and multithreaded workloads. Xen has almost no virtualization overhead for single-threaded workloads, but performs poorly with synchronized multithreaded workloads.

1. INTRODUCTION
Virtualization is a technology used to partition the resources of a physical computer into virtual machines (VMs) [14]. This is often done with the intent of increasing the utilization of a physical computer used to host VMs. Many workloads use only a part of a physical host’s resources, but must run in a dedicated OS because of system management or security requirements. These types of workloads can be encapsulated in a VM, and multiple VMs can run on a single physical host.

Symmetric multiprocessing (SMP) is a way to combine the resources of multiple CPUs in a single host system. In an SMP system, multiple CPUs share access to main memory and execute concurrently, any process executing on the system can execute on any CPU, and the OS controls which CPU a process executes on. SMP is becoming increasingly important because of multi-core CPUs that combine multiple processor cores inside a single physical CPU. SMP is supported by most modern operating systems, and currently is the most common way to use the resources of multicore CPUs in general purpose computers.

In some ways, virtualization and SMP are technologies with opposite goals. Virtualization divides the resources of a single host among multiple VMs, while SMP combines the resources of multiple CPUs in a single host. However, combining these two technologies to create an SMP VM—that is, a virtual machine with multiple virtual CPUs (VCPUs)—is often useful.

Although virtualization is an idea that has been around for a long time [12], the original design of the x86 architecture made it difficult to implement virtualization on x86 systems [13]. Currently three different virtualization strategies are commonly used: binary translation, paravirtualization, and hardware-assisted virtualization. Binary translation rewrites x86 binary code to make it safe to run in a VM. This was the first virtualization technique that VMware used [3], and it is still the default choice for many guest operating systems running in ESX 3.5. Paravirtualization requires modifying the guest OS in order for it to support virtualization, and this technique was first implemented in Xen [7]. Finally, Intel and AMD have added virtualization extensions to the x86 architecture, and it is now possible to implement a VMM that does not use binary translation or paravirtualization [15]. VMware ESX 3.5 supports all three different modes of virtualization; while Xen makes extensive use of hardware-assisted virtualization to run guest OS that cannot be modified (such as Microsoft Windows). We used the binary translation mode of ESX, and the paravirtualization mode of Xen for all of our experiments because we believed that these were the most common use cases for the commercial product from VMware and the open source product from xen.org at the time our study was conducted.

VMware ESX and Xen are two of the leading virtualization systems for the x86 architecture, and they both allow for SMP virtualization. Implementing SMP virtualization is difficult because the two technologies have different goals, and virtualization in particular can conflict with the expected behavior of an SMP system. In this paper we analyze the performance of SMP virtualization in VMware ESX and in Xen. For our analysis we used two CPU intensive benchmarks from two different SPEC benchmark suites: 445.gobmk from CPU2006 for single-threaded workloads, and 332.ammp from OMP2001 for multithreaded workloads. We found that ESX had a slowdown of up to 12% for the 445.gobmk single-threaded benchmark, and a slowdown of up to 6% for the 332.ammp multi-threaded benchmark, when VMs were
configured with extra VCPUs. Xen in contrast had excellent performance for the 445.gobmk single-threaded benchmark, but did poorly with the 332.ammp.m benchmark with a slowdown of over 1000% in the worst case.

The rest of this paper is organized as follows: Section 2 describes our evaluation methodology, Section 3 describes the results for VMware ESX, Section 4 describes the results for Xen, Section 5 describes related work, and Section 6 provides a conclusion.

2. EVALUATION METHODOLOGY

The purpose of SMP is to provide additional CPU resources in order to execute multiple threads simultaneously. We focused our evaluation on CPU resources in order to analyze system performance when CPU resources are the limiting factor. Furthermore, we selected benchmarks that have very low virtualization overhead and execute almost entirely in user mode. This allowed our evaluation to focus on the relative performance of the SMP virtualization rather than virtualization as a whole.

We used selected benchmarks from two different SPEC benchmark suites designed to measure the relative performance of a system’s CPU and memory. The 445.gobmk benchmark from the SPEC CPU2006 suite was selected because of its low memory requirement which allowed us to run multiple instances on multiple different VMs all on a single physical host. We used 445.gobmk to measure the performance of independent single-threaded workloads.

Accurately evaluating the performance of multi-core and multi-threaded applications is more difficult than evaluating the performance of single-threaded applications. Even executing multiple instances of non-cooperative single-threaded benchmarks is difficult because benchmark completion time may vary [18]. The benchmarks we ran were executed simultaneously on up to 8 VMs at a time. In order to mitigate the problem of varying benchmark completion time changing the load on the host, we executed each benchmark configuration 4 times, and averaged the execution time of the first 3 iterations of the benchmark execution. This ensured that the load on the host was consistent for each benchmark iteration; the last iteration was not included because benchmark completion time could vary. This helped to ensure that the host system had a consistent load for the execution results we measured.

However, this also increased the complexity of our experiments by requiring measurement of multiple iterations of each benchmark across multiple VMs. Although we limited our experiments to a single benchmark in the SPEC CPU 2006 suite, we believe this is representative for measuring relative performance of differing numbers of VMs and VCPUs per VM on a single host.

For cooperative multi-threaded workloads, we selected a benchmark from the OMP2001 benchmark suite. Our host system had 8 logical processors (CPU cores) and as a result the maximum theoretical speed-up for a multi-threaded application was 8. However, the benchmarks in the OMP2001 suite do not scale linearly [6] with the number of processors and have a maximum speedup of less than 8. We evaluated several of the OMP2001 benchmarks executing natively on our host system and found that 332.ammp.m had a speedup of 6.5 when scaling from 1 to 8 CPUs. We wanted a benchmark that was able to scale reasonably well with the number of processors, while still requiring synchronization between threads. The 332.ammp.m benchmark met these requirements and we used it for evaluating the performance of cooperative multithreaded workloads.

Our host system was a Dell Poweredge 1950 with dual quad-core Intel Xeon X5355 2.66 GHz CPUs and 16 GB of RAM. This provided 8 logical CPUs in our host system which we used as the maximum number of VCPUs per VM in our experiments with Xen. Our system configurations are summarized in Table 1. We used Ubuntu server 7.04 32-bit version as our guest OS; however, for Xen guests we used a modified Linux kernel that was distributed with Xen and included support for paravirtualization. ESX 3.5 is limited to a maximum of 4 VCPUs per VM, which was the maximum value we tested for ESX. All results were collected by measuring benchmark execution time inside the VM, and the experiments run in ESX used VMware tools to synchronize the guest OS clock with the host. Benchmark execution times ranged from 10 minutes to 14 hours depending on the experiment. Because we evaluated different modes of virtualization for ESX and Xen, the absolute performance numbers we obtained are not directly comparable. However, our goal was only to evaluate the relative performance of each system with varying numbers of VCPUs per VM to analyze how the implementations of virtual SMP scaled. In order to focus on relative performance, we normalized the results for each benchmark to the execution time of a single VM with 1 VCPU. Each graph shows results normalized to the execution time of a 1 VCPU per VM configuration, and in all cases lower values are better. The versions of the virtualization software tested were ESX 3.5.0 and Xen 3.2.0 compiled from source code from xen.org.

3. ESX

VMware ESX can use a technique called binary translation to virtualize CPU resources in an x86 system [5]. This technique requires dynamically rewriting the binary code executing in a VM to ensure that ESX is able to preempt the guest OS executing in a VM when necessary. It also requires additional memory for each VCPU in a running VM that is used as virtualization overhead.

When combining multiple VCPUs in an SMP VM, ESX uses a technique called relaxed co-scheduling to determine how to schedule the execution of VCPUs on physical CPUs. Co-

<table>
<thead>
<tr>
<th>System</th>
<th>PowerEdge 1950 Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2 x Quad Core Intel Xeon X5355</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB RAM</td>
</tr>
<tr>
<td>Disk drive</td>
<td>4 x 146 GB 10,000 RPM SAS drive</td>
</tr>
<tr>
<td>RAM per VM</td>
<td>1.5 GB</td>
</tr>
<tr>
<td>ESX version</td>
<td>VMware ESX Server 3.5.0</td>
</tr>
<tr>
<td>ESX guest</td>
<td>Linux 2.6.20-15</td>
</tr>
<tr>
<td>Xen version</td>
<td>Xen 3.2.0 (dom0 Linux 2.6.18.8-xen)</td>
</tr>
<tr>
<td>Xen guest</td>
<td>Linux 2.6.18.8</td>
</tr>
</tbody>
</table>
scheduling requires that all VCPUs associated with a VM be scheduled simultaneously in order for that VM to run. However, ESX has several optimizations to improve performance over a naive implementation of co-scheduling, which would require even idle VCPUs in a VM to execute. First, ESX is able to detect if a VCPU is executing an idle loop, and in this case ESX does not schedule an idle VCPU to run nor require it to be co-scheduled for active VCPUs to run [2]. Second, ESX uses a technique called relaxed co-scheduling that helps prevent requiring physical CPUs from being idle in order to start running VCPUs in an SMP system [4].

Our first experiment measured the performance of single-threaded workloads executing in ESX. Figure 1 shows the results for a single VM on the host executing the 445.gobmk benchmark with varying numbers of VCPUs per VM. This graph is normalized to the execution time of the 1-VCPU configuration, and adding additional VCPUs to a VM introduces a slight performance penalty. Going from 1 VCPU to 2 VCPUs results in a 0.1% slowdown, and going from 1 VCPU to 4 VCPUs causes a 0.6% slowdown. While this is a small performance overhead, it was consistent across multiple iterations of benchmarking; in each test the fastest execution time was within 0.05% of the slowest execution time. In this test the host system was underutilized, as it had to schedule at most 4 VCPUs on 8 physical CPUs. Furthermore, the 445.gobmk benchmark workload would only use 100% of a single VCPU's resources, and other VCPUs would be idle. However, it is worth observing that simply configuring a VM with additional VCPUs results in a small but measurable performance overhead even on an underutilized ESX host system.

**Figure 1: Relative execution time of 1 VM executing 1 instance of 445.gobmk on ESX**

The next benchmark measured the performance of a single-threaded workload running in a resource constrained system. For this experiment, 8 VMs simultaneously executed the 445.gobmk benchmark on a host system with 8 logical processors; the results are shown in Figure 2. The benchmark was run with 1, 2, and 4 VCPUs per VM, and the relative execution time was recorded. The 2 VCPU configuration had a 2% slowdown compared to the 1 VCPU configuration, and the 4 VCPU configuration had a 12% slowdown compared to the 1 VCPU configuration. This slowdown shows a noticeable amount of overhead; however, it is far less than a naive implementation of co-scheduling — which would use CPU resources to schedule idle VCPUs — might experience. If ESX used a strict co-scheduling implementation the 2 VCPU configuration would have a 200% slowdown, and the 4 VCPU configuration would have a 400% slowdown. ESX is able to deschedule halted VCPUs, and it also detects and deschedules VCPUs where the guest OS is executing an idle loop. ESX performs well when scheduling single-threaded applications, but the host system does experience performance degradation when allocating unused VCPUs in a resource-constrained system. These results agree with the recommendation that VMware provides: do not configure VMs with unused VCPUs [1, 2].

**Figure 2: Relative execution time of 8 VMs each executing 1 instance of 445.gobmk on ESX**

The next set of benchmarks show results for a cooperative multithreaded workload executing in ESX on an underutilized host system. Figure 3 shows the results for a single VM executing the 332.ammp_m benchmark from the OMP2001 benchmark suite. Unlike the 445.gobmk benchmark, 332.ammp_m is multithreaded and can have improved performance when additional processors are available. Here the speedup is nearly linear with the number of VCPUs available, the 2 VCPU configuration’s runtime is 52% of the 1 VCPU configuration’s runtime, and the 4 VCPU configuration’s runtime is 28% of the 1 VCPU configuration’s runtime. This speedup is largely a function of the design of the 332.ammp_m benchmark; the benchmark determines how many threads to execute based on the number of CPUs available when it begins execution. However, the fact that an SMP VM is able to utilize these resources effectively shows that ESX does a good job of scheduling CPU resources on a host that is not resource-constrained.

**Figure 3: Relative execution time of 1 VM executing 332.ammp_m on ESX**

The final set of benchmark results shown for ESX are for a cooperative multithreaded workload executing on an overcommitted host. Figure 4 shows the performance of 8 VMs each simultaneously executing the 332.ammp_m benchmark.
In this case the 1 VCPU configuration will use all available host CPU resources, and adding additional VCPUs will result in more runnable threads in the system than available cores. However, these results show a very reasonable level of CPU overhead: the 2 VCPU configuration is 3\% slower than the 1 VCPU configuration and the 4 VCPU configuration is 6\% slower than the 1 VCPU configuration. This experiment demonstrates the advantage of ESX co-scheduling VCPU execution. The 332.ammp benchmark has a significant amount of communication between threads, and uses spinlocks for fast synchronization. By co-scheduling VCPUs, ESX provides good performance for this workload, and the resulting slowdown is only caused by additional virtualization overhead, not synchronization problems in the guest OS executing inside a VM.

The credit scheduler allows for up to 32 VCPUs per VM; however, we limited our experiments to 8 VCPUs per VM to match the maximum number of logical CPUs in our host system. The credit scheduler also schedules each VCPU independently; unlike ESX it does not attempt to synchronize execution of multiple VCPUs in a VM.

In Xen, unused VCPUs did not incur measurable overhead for the experiments we ran; benchmark execution time for a single VM executing the 445.gobmk was unchanged regardless of the number of VCPUs assigned to the VM. This contrasts with the results in ESX, where a single VM had a small but measurable performance degradation as more VCPUs were added. Although we measured the execution time of a single VM with varying numbers of VCPUs, we did not observe any consistent variation in execution time and so the results are not shown in a graph. Idle VCPUs have the lowest priority in Xen, and they introduce very little accounting cost for the credit scheduler.

The first experiment with Xen that shows variation in results is for 8 VMs executing simultaneously, with each executing a single instance of the 445.gobmk single-threaded benchmark. The 2 and 4 VCPU configurations have a 0.15\% slowdown compared to the 1 VCPU configuration, and the 8 VCPU configuration has a 0.4\% slowdown compared to the 1 VCPU configuration. Scheduling VCPUs in Xen is very similar to scheduling threads in Linux; because of this we included results for an additional experiment in our measurement of Xen. We had a single VM with 8 VCPUs executing 8 instances of the 445.gobmk benchmark, and found that it had a slowdown of 1.5\% compared to 8 VMs each with 1 VCPU. The results of these experiments are shown in Figure 5.
again demonstrate that additional VCPUs in Xen introduce minimal performance overhead. Also included for comparison was a single VM with 1 VCPU executing 8 instances of 445.gobmk. This means that there are only 8 threads executing on the host, instead of 64 as in the other configurations. However, these 8 threads are confined to executing on a single VCPU, so the host system can use at most 100% of a 1 CPU. The runtime for a single VM with 1 VCPU executing 8 threads is 95% of the runtime for 8 VMs, each with 1 VCPU with each executing 8 threads. The performance improvement likely comes from reduced utilization of host memory bandwidth and shared L2 cache. However, Xen demonstrates consistent performance, even when heavily overcommitted.

As noted earlier, the 332.ammp.in benchmark requires a significant amount of communication between threads cooperating on executing the benchmark. When the benchmark begins execution it sees how many processors are available to its OS, and it spawns an equal number of threads and divides its work between them. In order to facilitate fast synchronization, the threads use spinlocks to synchronize shared data. When a thread is waiting for a spinlock, it continues to use CPU resources until the lock is free. On a physical host this allows for very fast synchronization because no context switch is required and the thread can resume execution immediately after the lock is freed, rather than having to be rescheduled by the OS. In a physical system where these resources would otherwise be idle this is an efficient programming model for highly synchronized workloads. ESX uses co-scheduling for VCPUs in an SMP VM and this allows it to execute synchronized multithreaded workloads in a way that is very similar to how they would execute on a physical system.

However, Xen does not synchronize the execution of VCPUs, resulting in significant slowdown for workloads that require extensive synchronization between threads. When a thread executing on a VCPU is spinning (that is, executing a tight loop waiting for a lock to be released) it will not yield the physical CPU that it is executing on. However, if the thread that is holding the lock is executing on a VCPU that has been descheduled, then the VCPU with the thread that it is spinning may waste its entire timeslice. This is what causes the significant performance degradation shown in our results.

5. RELATED WORK

Optimizing the performance of SMP virtualization is a topic of ongoing research, and there are several proposals for new techniques for implementing SMP virtualization. Wells, et
Uhlig, et al [17, 16] have proposed modifying the guest OS so that it can notify the VMM when it will acquire a kernel level lock. Their work focused on preventing preemption of a VCPU holding a kernel level lock. However, it does not address problems with synchronization for application-level locks such as those used in the OMP2001 benchmark suite. In addition, their approach is focused on applications that require relatively limited synchronization, not highly synchronized workloads such as that in the 332.ammp.m benchmark.

Analysis of virtualization performance is also a topic of ongoing research, and as an open source project, Xen has been especially popular. Cherkasova, et al [8] have analyzed the performance of the different scheduler options in Xen with particular emphasis on the precision of resource allocation. They found that the credit scheduler used in Xen did not allocate CPU resources in accordance with configured values for some workloads. Ongaro, et al [11] analyzed the impact of the CPU scheduling on network I/O performance in Xen. They found that Xen favored allocating CPU resources to CPU-bound VMs rather than I/O-bound VMs, and that this could affect network bandwidth and latency in undesirable ways. Our work adds to the efforts to characterize the performance of virtualization scheduling algorithms by examining the performance of SMP VMs in ESX and Xen.

6. CONCLUSION

Virtualization is becoming an increasingly important technology in large part because it offers the promise of allowing more efficient use of computing resources. However, the behavior of a VM often differs significantly from a physical system, leading to performance degradation for applications running in a VM. Furthermore, virtualization is a rapidly evolving field, as hardware support continues to be added and adopted to bring the performance of virtualized systems closer to that of native execution. We analyzed the performance of ESX only for its most commonly used mode of virtualization, which is binary translation; however, there is ongoing work to extend paravirtualization features and make better use of improved hardware support for virtualization. We analyzed the performance of Xen using only paravirtualization, which is the virtualization technique that it introduced to the x86 architecture. However, Xen is also increasing adoption of hardware support for virtualization in order to provide support for a greater range of guest operating systems. The continuing innovation in virtualization systems is likely to bring changes to performance of SMP virtualization in the future.

We have found that two of the leading commercial virtualization platforms have very different scheduling implementations for SMP virtualization. SMP is the most common way to use the resources provided by multicore CPUs, and it is reasonable to expect that SMP will become increasingly important in the future. The co-scheduling approach used by VMware ESX leads to good overall performance, but it is limited in the number of VCPUs per VM, and it does experience noticeable overhead for using SMP with independent threads. Xen does not synchronize the execution of VCPUs, and while this provides excellent performance for independent threads, it can lead to significant performance degradation for workloads requiring synchronization.

7. REFERENCES


