Analysis of SMP VM CPU Scheduling

Gabriel Southern
George Mason University
gsouther@gmu.edu

Abstract

System virtualization allows resources of a single physical machine to be shared among multiple virtual machines (VMs). This can increase system flexibility, but it can also invalidate some assumption that the OS running in a VM has about the system it is executing on.

This paper surveys the implementation of symmetric multiprocessing (SMP) in a virtualized environment. It examines two proposals for optimizing the implementation of SMP in VMs, and it analyzes the implementation of SMP VMs for two popular virtualization systems: VMware ESX server and Xen. The benchmark results show that SMP VMs do experience greater overhead than uniprocessor VMs for computationally intensive workloads. However, optimizations in ESX server and Xen reduce the overhead below what it would be for the most naive implementation.

1. Introduction

System virtualization uses software to partition the resources of a single physical computer into multiple virtual machines (VMs). This technology was first developed when large, expensive mainframe systems dominated the marketplace. One of the earliest commercial virtualization implementations was the IBM VM/370 [6], which entered development in 1964 and is sold today as z/VM [8]. Popek and Goldbergs seminal 1974 paper [9] formally defined a virtual machine and the architecture requirements necessary to support virtualization.

Although system virtualization was popular in mainframe computers, it was not implemented in the low-cost commodity microprocessor systems that came to dominate the computing industry in the 1980s [11]. In particular, the Intel x86 architecture contained instructions that violated the requirements defined by Popek and Goldberg for implementing a “trap-and-emulate” virtualization system [10].

As computers using commodity microprocessors became more powerful, the development of operating systems became increasingly complex. Researchers at Stanford looked to virtualization to increase the efficiency of microprocessor based systems [3]. The company VMware was an outgrowth of this research, and produced efficient virtualization software for the x86 architecture [11].

VMware’s virtualization implementation used a technique called binary translation. While this provided good performance for many workloads, it did introduce some virtualization overhead which reduced system performance below that of the native system. An alternative approach called paravirtualization was proposed by researchers at the University of Cambridge [2]. This approach involved modifying the guest operating systems to optimize them for virtualization, without requiring any changes to applications running in the operating systems. They implemented their idea in the Xen hypervisor and released their implementation as an open-source product under the GPL license.

Today, VMware and Xen both provide high-performance virtualization software used by a variety of customers for server consolidation. This virtualization software is usually run on symmetric multiprocessor systems, and both VMware ESX server and Xen support multiprocessor VMs. However, the recommended best practice is to configure a VM with the number of virtual CPUs (VCPUs) that it will actively use, because SMP VMs introduce additional system performance overhead [17].

This paper surveys the implementation of symmetric multiprocessing (SMP) guest VMs for commodity virtualization systems. Section 2 provides background on the purpose and implementation of SMP. Section 3 surveys two proposals for optimizing scheduling for SMP virtual machines. Section 4 provides details about the implementation VMware ESX server. Section 5 provides an overview of Xen focused on its CPU scheduling design. Section 6 shows results from benchmarks run in SMP VMs on both ESX server and Xen, and discusses opportunties for future research. Section 7 summarizes the results and presents some conclusions.

2. Symmetric Multiprocessing

Multiprocessing describes a system where two or more processors are combined in a single system to share some system resources while enhancing system capabilities in some way [13]. Symmetric multiprocessing (SMP) is a special case of multiprocessing in which all processors behave identically and any processes in the system can execute on any processor.

Most modern operating systems provide support for the SMP system, and the kernel is typically more complex than a corresponding uniprocessor kernel. An SMP system can execute multiple independent processes simultaneously, which requires additional synchronization code for data structures to ensure consistency between multiple threads. For example, in Microsoft Windows, a uniprocessor kernel can disable interrupts at a relatively low cost in order to ensure that a series of instructions complete atomically. However, disabling interrupts on a multiple processors is more difficult, incurs a much greater cost, and is not an efficient use of system resources [12]. As a result, SMP systems require greater use of locks and other synchronization methods to ensure atomic access to shared data structures.

One type of lock commonly used in SMP systems is a spinlock [13], so named because the waiting process “spins” while waiting to acquire the lock. This is inefficient in a uniprocessor system, but in an SMP system it can be more efficient than performing a context switch if the lock is only going to be held for a short time. Because the operating system controls the system resources and processor scheduling, it can determine which locks are only held for a short time and thus when to use a spinlock rather than a context switch.

This is not the case, however, when the operating system runs as a guest OS inside a virtual machine. Instead of directly controlling system resources, the guest OS shares resources with other VMs, and the VMM retains ultimate control for scheduling system resources. This can potentially cause both performance and correctness problems. For example, a VCPU that is executing a spinlock may be scheduled on a physical CPU (PCPU), while the lock holder it is waiting on is not scheduled. The VCPU then would waste its entire time-slice waiting for the lock to be released. It would be much more efficient to de-schedule the VCPU that is spinning and schedule the lock-holder so that it can complete its work and release the lock. If there is a timeout while executing the spinlock, the system may also experience correctness problems because the OS may determine that portions of the system have failed when in fact the virtualization layer has simply changed the system timing characteristics.

One solution to this problem would be for the VMM to schedule all VCPUs in a VM simultaneously. However, while simple, this solution is inefficient, because physical CPUs will be idle if there are not enough available to schedule all VCPUs simultaneously. For example: consider a system with four PCPUs and two VMs. The first VM has four VCPUs and the second VM has only one VCPU. When the first VM is running, all four VCPUs must be scheduled, so the second VM must be de-scheduled. If some of the VCPUs in the first VM are simply executing the idle loop, this will be an inefficient use of system resources. Even more inefficient is when the second VM is scheduled and three VCPUs must be idle even if the first VM still has work to do.

3. Proposed Solutions

Scheduling SMP VMs is complicated primarily because of difficulties with lock-holder preemption. Some systems simply ignore the situation in the hope that lock-holder preemption will not be a problem. Others co-schedule the VCPUs in an SMP VM so that the VM has the same semantics as it would if it were executing on physical hardware. Other proposals include a software technique that tries to eliminate lock-holder preemption, and a hardware technique that tries to preempt VCPUs that are not performing useful work.

3.1 Avoiding Lock Holder Preemption

Uhlig, et al. [16] propose using a software technique they call “lock-holder preemption avoidance.” Instrumenting the Linux 2.4.20 kernel to trace kernel locks and measure system statistics under a variety of system workloads,they found that the likelihood of preempting a VCPU holding a lock ranged from 0.04% for a CPU-intensive workload to 39% for an I/O bound workload.

Two types of lock-holder preemption techniques were proposed: intrusive and non-intrusive. The intrusive technique required modifying the guest-OS kernel to have the OS indicate how long a lock should be held. A VCPU holding a lock would be guaranteed not to be preempted while holding the lock for the amount of time that it requested. If it had not released the lock by the end of the allotted time, then preemption would still be allowed.

Since the intrusive technique is not suitable for operating systems distributed in binary form, an alternative non-intrusive technique was also proposed. The non-intrusive technique divides system states into safe and unsafe preemption states. All kernel-level locks are guaranteed to be released before switching to user mode, so user-mode is part of the safe state. If the OS is executing the idle loop, it should also have released all locks before entering this state, so this is also a safe state. Some workloads such as CPU-intensive applications will almost always be in a safe state, while others will rarely be safe for preemption. To address the problem of VM workloads that are rarely safe for preemption, a special device driver can be loaded that will force the guest-OS into a safe state.

3.2 Hardware Support for Spin Management

An alternative approach proposed by Wells, et al. [22] is to add hardware support to processors to detect when a VCPU is not performing useful work, and to preempt the processor. The authors note that when a process is executing without making forward progress, system state will be largely unchanged. They propose adding a CPU counter that will check for k store instructions for every N committed instructions. With fewer than k stored instructions, the system decides the process is not making forward progress, preempts it, and schedules a new process.

An advantage of this proposal is that a VM can be configured with more VCPUs than there are PCPUs in the system and still sustain acceptable performance. The proposed implementation of this technique also adds hardware support to facilitate rapid switching between VCPUs by reserving a dedicated section of memory for VCPU state information.

They simulated the proposed hardware spin management solution was simulated using Virtutech Simics full-system simulator modeling a SunFire 6800 with UltraSPARC IIICu CPUs using the SPARC V9 ISA. The simulator was run with a variety of workloads and found that on average the proposed hardware support could achieve a 10-25% speedup over gang scheduling.

4. VMware ESX Server

VMware ESX Server is a popular commercial virtualization system for the x86 architecture that runs on the physical hardware with no lower level OS [18]. This type of virtualization software is sometimes called “bare-metal” virtualization because the VMM has direct control of the system’s computing resources. Other VMware products, such as VMware Workstation or VMware Server, run as applications in a host operating system and are called hosted virtualization products [20]. Having direct control of the system resources allows ESX server to reduce the virtualization overhead and provide high-performance virtualization of system resources running many virtual machines.

VMware’s virtualization products use a technique called binary translation to virtualize the x86 architecture. The virtualization technique used in mainframe systems is sometimes called “trap-and-emulate” [1]. With this technique, privileged instructions executed by a VM–ones that could change system state–would trigger a system trap, causing system execution to jump to the VMM. The VMM could then validate that the VM was allowed to perform the privileged operation, emulate it, and transfer control back to the VM. However, the x86 architecture did not originally support this type of virtualization, because some privileged operations executed in user mode might simply fail without causing a trap that would allow control to be transferred to the VMM. VMware’s binary translation virtualization technique solved this problem by having the VMM examine the binary instructions before they were executed, and dynamically rewrite sections of code that would try to execute a privileged instruction so that the VMM would maintain control of the system and emulate the effect of the instruction.

The primary reason that ESX server provides higher performance than hosted virtualization products such as VMware server is that the VMM has direct control of system resources rather than having to request them through a host operating system [21]. One of the areas this benefits is CPU scheduling. In ESX server, the VMM directly controls scheduling of VCPUs on PCPUs rather than having VCPUs execute as threads in a host OS that are scheduled based on the host OS CPU scheduling algorithm.

ESX server supports guest operating systems with one, two, or four VCPUs; however, VMware recommends not configuring a VM with more VCPUs than it will actively use [19, 17]. There are two reasons for this: first, each VCPU causes additional system load even when it is idle, and second, increasing the number of VCPUs in a VM limits the flexibility of the VMM in scheduling when and where the VCPUs execute.

ESX server uses co-scheduling for VCPUs in order to maintain correctness and high performance [19]. Using co-scheduling eliminates the problems with synchronizing locks between different VCPUs and does not require special hardware support or changes to the guest operating system. However, it can cause inefficiency in the scheduling because it can cause PCPUs to be idle while waiting to schedule VCPUs. ESX server does have one significant optimization to its scheduling algorithm for SMP VMs: It can detect when a VCPU is executing the idle loop, and idle VCPUs are not scheduled on PCPUs.

5. Xen

Xen is a virtualization system for the x86 architecture that followed a different design philosophy from VMware's binary translation focusing on reducing the virtualization overhead and providing high performance. While VMware is a commercial company, Xen is an opensource software product that grew out of research done at the University of Cambridge [2]. Many vendors contributed to the development of Xen, and offer it in commercially supported products, either standalone or integrated with an operating system. The source code for Xen remains freely available for download and anyone can contribute to its further development.

Xen uses a design called paravirtualization to optimize the virtualization system. Instead of supporting unmodified guest operating systems, Xen requires that the guest OS be modified to optimize the virtualization interface, while still supporting unmodified applications running in the guest OS. Xen took advantage of the fact that the x86 architecture supports four different privilege levels, called rings, but most operating systems only use ring 0, the most privileged ring used for kernel mode code, and ring 3, the least privileged ring used for user mode code. Xen modified the guest OS to use ring 1 for kernel mode code, and kept ring 3 for application mode code. This left ring 0 available for the Xen hypervisor, which allowed it to maintain overall control of the system. Xen also required some additional changes to the guest OS to optimize I/O and memory management. However, the total amount of effort required to modify a guest OS so that it can be used as a paravirtualized guest in Xen is still reasonable. The first implementation of Xen required modifying 2995 lines of Linux code, which was 1.36% of the total x86 code base for Linux [2].

Xen has continued to evolve since the original open source implementation was released and it is now in its third version. One of the most significant changes during this time was the addition of support for virtualization extensions added to the Intel and AMD processors. The original implementation of Xen only supported paravirtualized guest operating systems, but modern x86 processors with virtualization extensions can now support unmodified guest operating systems [7].

One of the design goals for Xen is the separation of policy from mechanism: Xen provides the framework for controlling system functionality, but the system configuration can be configured or extended by system administrators [4]. One of the areas where this is seen is in the CPU scheduling process. The basic functionality required for CPU scheduling is the ability to execute VCPUs on PCPUs; however, the precise method of accomplishing this is a policy decision.

As an open source product, Xen is designed to be extensible, and the CPU scheduling demonstrates this. Xen provides a relatively clean interface for implementing a new scheduling algorithm, and can be configured to choose between multiple different scheduling algorithms at system boot time. Xen currently supports two different scheduling algorthims: simple earliest deadline first (SEDF) and the credit scheduler. The credit scheduler is the newer scheduler and is the default in the latest version of Xen.

While VMware ESX server 3.x supports up to four VCPUs per VM, the open source Xen 3.x hypervisor supports up to 32 VCPUs per VM [23]. Commercially supported versions of Xen typically support fewer VCPUs per guest; for example, Citrix XenServer 4.1 supports up to eight VCPUs per guest [5].

The credit scheduler used with the open-source Xen 3.2 hypervisor does not attempt to address the possible problem of a lock-holding VCPU being preempted.

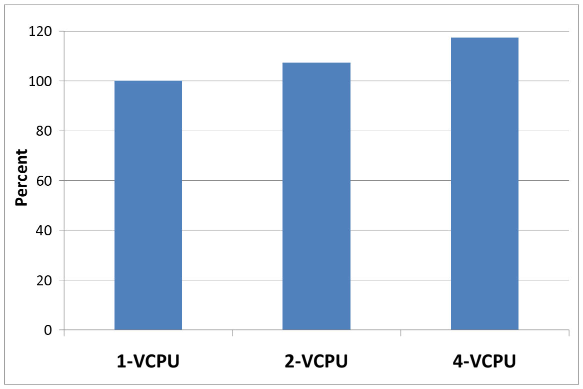

Figure 1. ESX server VMs running CPU2006 bzip2 benchmark. Results normalized to one VCPU case. Lower is better.

Instead, VCPUs are scheduled independently, and no effort is made to co-schedule the VCPUs in a VM[24]. This eliminates the inefficiencies that result from co-scheduling idle VCPUs or from keeping PCPUs idle in order to group VCPUs together. However, it introduces the potential for inefficiency due to preemption of a VCPU that is holding a lock.

6. Evaluation

To evaluate system performance I used two benchmarks suites from the Standard Performance and Evaluation Corporation (SPEC). I ran the benchmarks in VMs is VMware ESX server 3.5 and in the open source Xen 3.2 virtualization platform. The first benchmark suite used was SPEC CPU2006 [14], these benchmarks are single threaded applications designed to test the performance of systems memory and CPU subsystems. The other benchmarks used were SPEC OMP2001 [15], these also test CPU and memory performance, but use the OpenMP library to compile the programs as multithreaded applications. All benchmarks were executed on a Dell PowerEdge 1950 server with dual quad-core Xeon X5355 CPUs and 16GB of RAM.

6.1 ESX Server Results

The first benchmark run on ESX server executed ten iterations of the SPEC CPU2006 bzip2 benchmark simultaneously on eight VMs. The test was repeated three times: once with each VM configured with one VCPU, once with each VM configured with two VCPUs, and once with each VM configured with four VCPUs. In all tests each VM was configured with 1280 MB of RAM.

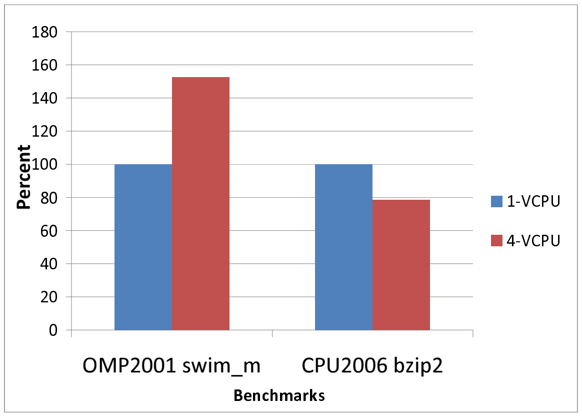

Figure 2. ESX server VMs running OMP2001 and CPU2006 bzip2 benchmark. Results normalized to one VCPU case. Lower is better.

The results from this benchmark are shown in figure 1. The one VCPU case is used as a baseline, and the other tests are shown as a precentage of the exectuion time of the one VCPU test. As increasing the number of VCPUs per VM increases the total execution time for the benchmark. This is because each additional VCPU causes additional system overhead because it requires timer events for each VCPU. However, the percentage increase in the time is far less than would be caused if ESX server had to co-schedule idle VMs. These results show that ESX server does a relatively good job of detecting and descheduling idle VCPUs when running compute intensive workloads.

The second test run on ESX server is a mix of single-threaded and multithreaded applications. The SPEC OMP2001 benchmarks require 2GB of RAM to execute. The server I was testing with has 16 GB of RAM; however, some RAM is used as overhead for each VM and each VCPU associated with each VM. This prevented me from simultaneously running eight VMs each executing the SPEC OMP2001 benchmark. Instead I ran five VMs with selected SPEC CPU2006 benchmarks and three VMs with OMP2001 benchmarks.

The results for each test are shown in figured 2, again normalized to the one VCPU case. The OMP2001 test used the swim_m benchmark, and the four VCPU VMs took approximately 52% longer to execute the tests than the one VCPU VMs. The four VCPU VMs incurred additional synchronization overhead because

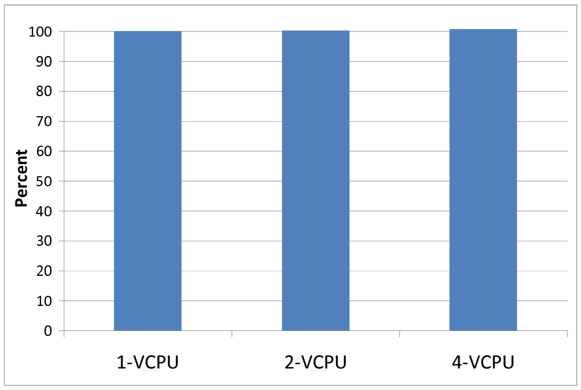

Figure 3. Xen VMs running CPU2006 bzip2 benchmark. Results normalized to one VCPU case. Lower is better.

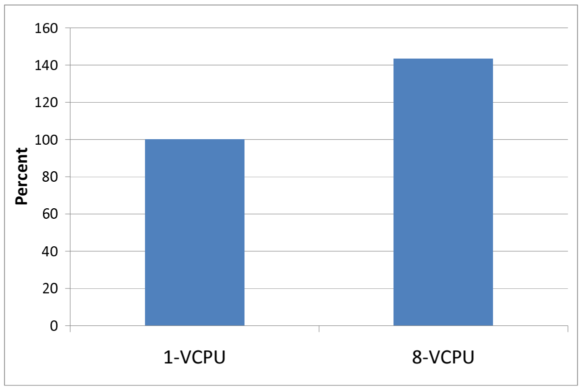

Figure 4. Xen VMs running the OMP2001 wupwise_m benchmark. Results normalized to one VCPU case. Lower is better.

they were executing multiple threads and this helped to cause the slowdown in execution speed.

For the single threaded CPU2006 benchmarks, the four VCPU VMs actually performed better than the one VCPU VMs. This was somewhat unexpected, because the earlier tests showed a reduction in system performance is expected in this instance. However, the explanation is that the entire system is overloaded and the VMs running the OMP2001 benchmarks are not able to accomplish as much work due to synchronization overhead. This reduces the amount of shared processor cache and memory bandwidth resources these VMs use, and leaves more available for use by the four VCPU VMs.

6.2 Xen Results

The first benchmark executed in Xen was the same as the first benchmark test run for ESX server: eight VMs each executing the SPEC CPU2006 bzip2 benchmark. The results are shown in figure 3 again normalized to the one VCPU execution time; however, the results were nearly identical for the three different configurations. The two VCPU execution time was 100.2% of the one VCPU execution time, and the four VCPU execution time was 101% of the one VCPU execution time.

These results help to verify that Xen is not doing any co-scheduling and is able to deschedule idle VCPUs. The paravirtualized interface used by Xen also reduces the overhead of having additional VCPUs for Xen.

The second test executed for Xen using the OMP2001 benchmarks was slightly different than the OMP2001 tests run in ESX server. The paravirutalized Xen guest operating systems caused a relatively low virtualization overhead and it was possible to configure eight VMs with enough memory to simultaneously execute one of the OMP2001 benchmarks. The benchmark that was selected was the wupwise_m benchmark. Xen also supports a greater number of VCPUs per VM. For this benchmark two configurations were tested: one running eight VMs each with one VCPU, and the other with eight VMs each with one VCPU. Eight VMs was selected as a maximum value to match the number of physical processors in the system.

The results indicate that the eight VCPU configuration takes approximately 43% longer to complete the benchmark tests than the one VCPU case. The earlier CPU2006 results had shown that there was relatively little virtualization overhead for single-threaded benchmarks running in Xen. However, these results show that for a multithreaded workload the need to synchronize multiple threads, combined with lack of co-scheduling, results in lower overall system performance.

6.3 Future Work

The results shown in this paper are preliminary and I am continuing to perform benchmark tests on the system in order to collect a more complete set of results and gather metrics for all benchmarks in the SPEC CPU2006 and OMP2001 suites. The benchmarks take a relatively long time to execute, on the order of hours for a single benchmark test to finish executing, and it is necessary to execute multiple iterations in order to obtain a statistically consistent average.

I am also working on creating a microbenchmark that will allow for easier characterization of the performance of an application that makes extensive use of spinlocks. I had begun collecting results using SPEC CPU2006 and OMP2001 benchmarks because they are well known established methods of testing performance of CPU subsystems. However, these applications contain very few spinlocks and so can not provide answers for some important questions about SMP VM scheduling.

Scheduling of VCPUs on PCPUs in an SMP VM remains an open areas of research. This paper exmained the impact of some scheduling algorithms for compute intensive workloads, but did not analyze the performance of I/O bound workloads to see what the impact of SMP scheduling on those workloads is. This paper has also provided some characterization of the performance of some popular compute intensive workloads, but these workloads contain very few spinlocks. Testing ESX server and Xen with workloads that use spinlocks remains an area of future work and offers the opportunity to provide a fuller characterization of the performance of these virtualization systems.

7. Summary and Conclusion

Scheduling for SMP VMs is a challenging problem, because the virtualization system changes the behavior of the underlying system so that it does not behave identically to physical hardware. When an operating system is running on physical hardware it has complete control of system resources and can expect certain predictable levels of system behavior. In SMP systems the operating system expects that all processors are executing simultaneously, and overall system performance can be optimized by using a spinlock in some situations.

Virtualization changes the underlying system behavior, and while a VM is similar to a physical machines, some system behavior can be different. In the case of SMP scheduling it is possible for the assumptions that the guest OS has about the system behavior to be invalid because all VCPUs in a VM may not be scheduled simultaneously.

This paper surveyed two proposals for optimizing the scheduling of SMP VMs, and also looked at the implementation of SMP VMs for two popular virtualization systems. One proposal suggested using software to ensure that a VCPU in a guest was not holding any synchronization locks before being preemepted. The other suggested adding hardware support to detect when a VCPU was running without performing useful work, as it would when using a spinlock to wait for an event on a VCPU that was not currently scheduled. Neither of these approaches have been implemented in commercial virtualization systems, but they are interesting ideas to consider for the future as SMP VMs become more prevelant.

The two virtualization systems that were studied were VMware ESX server and the open source implementation of Xen. ESX server is a popular commercial virtualization product and it has a conservative approach for SMP VM scheduling. It uses co-scheduling to ensure that the assumptions the guest OS has about system behavior will not be violated when running as a VM. However, ESX server does apply one important optimization, it is able to detect and deschedule idle VMs, which reduces system overhead and increases scheduling flexibility. Testing with a single threaded compute intensive workload shows a relatively low performance penalty for using SMP VMs. Testing with a multithreaded workload demonstrated a higher performance penalty; however, the VMware recommends against running a system with overcommitted CPU resources. Additional testing with custom written synchronization code is needed to futher characterize system performance.

As an opensource product that uses paravirtualiztion, Xen is able to use more aggressive scheduling techniques that can lead to higher system performance. The current default scheduler in Xen is the credit scheduler, and this scheduler does not attept to co-schedule VCPUs. This allows the system to optimize performance for compute intesive workloads and results in negligible performance degradation when configuring VMs with unused VCPUs. For multithreaded workloads a uniprocessor VM can outperform an SMP VM if the host system is overcommitted. This is caused by additional system performance overhead introduced by running a multithreaded application. A system with free resources would benefit from parallelizing the workload, but on an overcommited system this causes additional system overhead that reduces overall system performance.

Today the leading virtualization platform vendors recommend configuring VMs with only the number of VCPUs that will be in active use based on the system workload. However, the computing industry as a whole is moving to greater use of thread-level parralellism and this will eventually require configuring VMs with more VCPUs for the average system configuration. The SMP scheduling algorithms used by ESX server and by Xen have a low overhead for single-threaded compute intensive workloads. However, they do not scale as well for multi-threaded worloads on a system with overcommitted CPU resources. Some proposals have been made to improve the performance of SMP VM scheduling, but as of today they have not been implemented in the leading server consolidation virtualization platforms, and there will continue to be opportunites for research to improve on what is implemented in systems today.

References

[1] ADAMS, K., AND AGESEN, O. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS* (2006), ACM, pp. 2–13.

[2] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (New York, Oct. 19–22 2003), vol. 37, 5 of *Operating Systems Review*, ACM Press, pp. 164–177.

[3] BUGNION, E., DEVINE, S., AND ROSENBLUM, M. Disco: Running commodity operating systems on scalable multiprocessors. In *Proceedings of the 16th Symposium on Operating Systems Principles (SOSP-97)* (New York, Oct. 5–8 1997), vol. 31,5 of *Operating Systems Review*, ACM Press, pp. 143–156.

[4] CHISNALL, D. *The Definitive Guide to the Xen Hypervisor*. Prentice Hall, 2007.

[5] CITRIX. Citrix XenServer Product Overview. `http://www.citrixxenserver.com/Documents/XenServer41ProductOverview.pdf`, accessed: 04/29/08.

[6] CREASY, R. J. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development 25*, 5 (1981), 483–490.

[7] DONG, Y., LI, S., MALLICK, A., NAKAJIM, J., TIAN, K., XU, X., YANG, F., AND YU, W. Extending Xen with Intel virtualization technology. *Intel Technology Journal 10*, 3 (Aug. 2006), 193–203.

[8] PARZIALE, L., ALVES, E., DOW, E., EGELER, K., HERNE, J., JORDAN, C., NAVEEN, E., PATTABHIRAMAN, M., AND SMITH, K. *Introduction to the New Mainframe: z/VM Basics*. IBM International Technical Support Organization, 2007.

[9] POPEK, G. J., AND GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. *Communications of the ACM 17*, 7 (July 1974), 412–421.

[10] ROBIN, J. S., AND IRVINE, C. E. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In *Proceedings of the Ninth USENIX Security Symposium, August 14–17, 2000, Denver, Colorado* (pub-USENIX:adr), USENIX, Ed., USENIX.

[11] ROSENBLUM, M., AND GARFINKEL, T. Virtual machine monitors: Current technology and future trends. *IEEE Computer 38*, 5 (2005), 39–47.

[12] RUSSINOVICH, M., AND SOLOMON, D. *Microsoft Windows Internals*. Microsoft Press, 2004.

[13] SILBERSCHATZ, A., GALVIN, P., AND GANGE, G. *Operating System Concepts*. John Wiley & Sons, Inc., 2005.

[14] STANDARD PERFORMANCE EVALUATION CORPORATION. SPEC CPU2006. `http://www.spec.org/osg/cpu2006`, accessed: 04/29/08.

[15] STANDARD PERFORMANCE EVALUATION CORPORATION. SPEC OMP. `http://www.spec.org/omp`, accessed: 04/29/08.

[16] UHLIG, V., LEVASSEUR, J., SKOGLUND, E., AND DANNOWSKI, U. Towards scalable multiprocessor virtual machines. In *Virtual Machine Research and Technology Symposium* (2004), USENIX, pp. 43–56.

[17] VMWARE. Best Practices Using VMware Virtual SMP. `http://www.vmware.com/pdf/vsmp_best_practices.pdf`, 2005.

[18] VMWARE. Building the Virtualized Enterprise with VMware Infrastructure. `http://www.vmware.com/pdf/vmware_infrastructure_wp.pdf`, 2007.

[19] VMWARE. Performance Tuning Best Practices for ESX Server 3. `http://www.vmware.com/pdf/vi_performance_tuning.pdf`, 2007.

[20] VMWARE. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. `http://www.vmware.com/resources/techresources/1008`, 2007.

[21] VMWARE. Virtualization Overview. `http://www.vmware.com/pdf/virtualization.pdf`, 2007.

[22] WELLS, P., CHAKRABORTY, K., AND SOHI, G. Hardware support for spin management in overcommitted virtual machines. In *Proceedings of the 15th International Conference on Parallel Architecture and Compilation Techniques (15th PACT'06)* (Seattle, Washington, USA, Sept. 2006), ACM, pp. 124–133.

[23] XEN. Xen users' manual. `http://bits.xensource.com/Xen/docs/user.pdf`, 2007.

[24] XEN. Xen source code: sched_sedf.c. `http://lxr.xensource.com/lxr/source/xen/common/sched_sedf.c`, accessd: 04/29/08.