Co-scheduling SMP VMs in VMware ESX Server

Background

VMware ESX Server efficiently manages a mix of uniprocessor and multiprocessor VMs, providing a rich set of controls for specifing both absolute and relative VM execution rates. For general information on cpu scheduling controls and other resource management topics, please see the official VMware Resource Management Guide.

For a multiprocessor VM (also known as an "SMP VM"), it is important to present the guest OS and applications executing within the VM with the illusion that they are running on a dedicated physical multiprocessor. ESX Server faithfully implements this illusion by supporting near-synchronous coscheduling of the virtual CPUs within a single multiprocessor VM.

The term "coscheduling" refers to a technique used in concurrent systems for scheduling related processes to run on different processors at the same time. This approach, alternately referred to as "gang scheduling", had historically been applied to running high-performance parallel applications, such as scientific computations. VMware ESX Server pioneered a form of coscheduling that is optimized for running SMP VMs efficiently.

Motivation

An operating system generally assumes that all of the processors it manages run at approximately the same rate. This is certainly true in non-virtualized environments, where the OS manages physical processor hardware. However, in a virtualized environment, the processors managed by a guest OS are actually virtual cpu abstractions scheduled by the hypervisor, which time-slices physical processors across multiple VMs.

At any particular point in time, each virtual cpu (VCPU) may be scheduled, descheduled, preempted, or blocked waiting for some event. Without coscheduling, the VCPUs associated with an SMP VM would be scheduled independently, breaking the guest's assumptions regarding uniform progress. We use the term "skew" to refer to the difference in execution rates between two or more VCPUs associated with an SMP VM.

Inter-VCPU skew violates the assumptions of guest software. Non-trivial skew can result in severe performance problems, and may even induce failures when the guest expects inter-VCPU operations to complete quickly. Let's first consider the performance implications of skew. Guest OS kernels typically use spin locks for interprocessor synchronization. If the VCPU currently holding a lock is descheduled, then the other VCPUs in the same VM will waste time busy-waiting until the lock is released. Similar performance problems can also occur in multi-threaded

user-mode applications, which may also synchronize using locks or barriers. Unequal VCPU progress will also confuse the guest OS cpu scheduler, which attempts to balance load across VCPUs.

An extreme form of this performance problem may also lead to correctness issues. For example, a guest kernel may perform inter-processor operations, such as TLB shootdowns, that are expected to complete quickly on physical hardware (e.g. several microseconds). The guest OS may timeout if it finds that such operations have not completed after an unreasonably long period of time (e.g. several milliseconds). Without coscheduling, we have observed this behavior in practice for several different guest operating systems, including Windows BSODs, and Linux kernel panics.

Strict coscheduling in ESX Server 2.x

VMware introduced support for running SMP VMs with the release of ESX Server 2 in 2003. ESX Server 2.x implemented coscheduling using an approach based on skew detection and enforcement.

The ESX scheduler maintains a fine-grained cumulative skew value for each VCPU within an SMP VM. A VCPU is considered to be making progress when it is running or idling. A VCPU's skew increases when it is not making progress while at least one of its sibling VCPUs is making progress. A VCPU is considered to be "skewed" if its cumulative skew value exceeds a configurable threshold, typically a few milliseconds.

Once any VCPU is skewed, all of its sibling VCPUs within the same SMP VM are forcibly descheduled ("co-stopped") to prevent additional skew. After a VM has been co-stopped, the next time any VCPU is scheduled, all of its sibling VCPUs must also be scheduled ("co-started"). This approach is called "strict" coscheduling, since all VCPUs must be scheduled simultaneously after skew has been detected.

In some situations, such as when the physical machine has few cores, and is running a mix of UP and SMP VMs, coscheduling may incur "fragmentation" overhead. For example, consider an ESX Server with two physical cores running one dual-VCPU VM and one single-VCPU VM. When the UP VM is running, the scheduler cannot use the remaining physical core to run just one of the SMP VM's two VCPUs. This effect is typically negligible in systems with larger numbers of cores (or with hyperthreading enabled), due to the increased flexibility available when mapping VCPUs to hardware execution contexts.

Note that a VCPU executing in the guest OS idle loop can be descheduled without affecting coscheduling, since the guest OS can't tell the difference. In other words, an idle VCPU does not accumulate skew, and is treated as if it were running for coscheduling purposes. This optimization ensures that idle guest VCPUs don't waste physical processor resources, which can instead be allocated to other VMs. For example, an ESX Server with two physical cores may be running one VCPU each from two different VMs, if their sibling VCPUs are idling, without incurring any coscheduling overhead. Similarly, in the fragmentation example above, if one of the SMP VM's VCPU is idling, then there will be no coscheduling fragmentation, since its sibling VCPU can be scheduled concurrently with the UP VM.

Relaxed coscheduling in ESX Server 3.x

The coscheduling algorithm employed by the ESX scheduler was significantly enhanced with the release of ESX Server 3 in 2006. The basic coscheduling approach is still based on skew detection and enforcement.

However, instead of requiring all VCPUs to be co-started, only those VCPUs that are skewed must be co-started. This ensures that when any VCPU is scheduled, all other VCPUs that are "behind" will also be scheduled, reducing skew. This approach is called "relaxed" coscheduling, since only a subset of a VM's VCPUs must be scheduled simultaneously after skew has been detected.

To be more precise, suppose an SMP VM consists of multiple VCPUs, including VCPUs A, B, and C. Suppose VCPU A is skewed, but VCPUs B and C are not skewed. Since VCPU A is skewed, VCPU B can be scheduled to run only if VCPU A is also co-started. This ensures that the skew between A and B will be reduced. But note that VCPU C need not be co-started to run VCPU B. As an optimization, the ESX scheduler will still try to co-start VCPU C opportunistically, but will not require this as a precondition for running VCPU B.

Relaxed coscheduling significantly reduces the possibility of coscheduling fragmentation, improving overall processor utilization.

Conclusions

ESX Server employs sophisticated cpu scheduling algorithms that enforce rate-based quality-of-service for both uniprocessor and multiprocessor VMs. For multiprocessor VMs, coscheduling techniques ensure that virtual CPUs make uniform progress, faithfully implementing the illusion that the VM is running on dedicated multiprocessor hardware, ensuring efficient execution of guest software. Optimizations such as relaxed coscheduling and descheduling idle VCPUs provide a high-performance execution environment that efficiently utilizes physical host resources.

Appendix: ESX Server coscheduling statistics

ESX Server 3.x exports statistics related to the coscheduling behavior of multiprocessor VMs. The "esxtop" utility can be used to examine these statistics on a live ESX system.

The %CSTP column in the CPU statistics panel shows the fraction of time the VCPUs of a VM spent in the "co-stopped" state, waiting to be "co-started". This gives an indication of the coscheduling overhead incurred by the VM. If this value is low, then any performance problems should be attributed to other issues, and not to the coscheduling of the VM's virtual cpus.