Timekeeping in VMware Virtual Machines

VMware® ESX 3.5/ESXi 3.5, VMware Workstation 6.5

Because virtual machines work by time-sharing host physical hardware, a virtual machine cannot exactly duplicate the timing behavior of a physical machine. VMware virtual machines use several techniques to minimize and conceal differences in timing behavior, but the differences can still sometimes cause timekeeping inaccuracies and other problems in software running in a virtual machine. This information guide describes how timekeeping hardware works in physical machines, how typical guest operating systems use this hardware to keep time, and how VMware products virtualize the hardware.

This paper is intended for partners, resellers, and advanced system administrators who are deploying VMware products and need a deep understanding of the issues that arise in keeping accurate time in virtual machines. The VMware knowledge base contains additional and more frequently updated information, including best practices to configure specific guest operating system versions for the most accurate timekeeping, as well as recipes for diagnosing and working around known issues in specific versions of VMware products.

This document includes the following topics:

Timekeeping Basics

Computer operating systems typically measure the passage of time in one of two ways.

- Tick counting—The operating system sets up a hardware device to interrupt periodically at a known rate, such as 100 times per second. The operating system then handles these interrupts (called ticks) and keeps a count to determine how much time has passed.

- Tickless timekeeping—A hardware device keeps a count of the number of time units that have passed since the system booted, and the operating system simply reads the counter when needed. Tickless timekeeping has several advantages. In particular, it does not keep the CPU busy handling interrupts, and it can keep time at a finer granularity. However, tickless timekeeping is practical only on machines that provide a suitable hardware counter. The counter must run at a constant rate, be reasonably fast to read, and either never overflow or overflow infrequently enough that the operating system can reliably extend its range by detecting and counting the overflows.

Besides measuring the passage of time, operating systems are also called on to keep track of the absolute time, often called wall-clock time. Generally, when an operating system starts up, it reads the initial wall-clock time to the nearest second from the computer's battery-backed real time clock or queries a network time server to obtain a more precise and accurate time value. It then uses one of the methods described above to measure the passage of time from that point. In addition, to correct for long-term drift and other errors in the measurement, the operating system may include a daemon that runs periodically to check the clock against a network time server and make adjustments to its value and running rate.

Tick Counting

Many PC-based operating systems use tick counting to keep time. Unfortunately, supporting this form of timekeeping accurately in a virtual machine is difficult.

Virtual machines share their underlying hardware with the host operating system (or on VMware ESX, the VMkernel). Other applications and other virtual machines may also be running on the same host machine. Thus, at the moment a virtual machine should generate a virtual timer interrupt, it may not actually be running. In fact, the virtual machine may not get a chance to run again until it has accumulated a backlog of many timer interrupts. In addition, even a running virtual machine can sometimes be late in delivering virtual timer interrupts. The virtual machine checks for pending virtual timer interrupts only at certain points, such as when the underlying hardware receives a physical timer interrupt. Many host operating systems do not provide a way for the virtual machine to request a physical timer interrupt at a precisely specified time.

Because the guest operating system keeps time by counting interrupts, time as measured by the guest operating system falls behind real time whenever there is a timer interrupt backlog. A VMware virtual machine deals with this problem by keeping track of the current timer interrupt backlog and delivering timer interrupts at a higher rate whenever the backlog grows too large, in order to catch up. Catching up is made more difficult by the fact that a new timer interrupt should not be generated until the guest operating system has fully handled the previous one. Otherwise, the guest operating system may fail to see the next interrupt as a separate event and miss counting it. This phenomenon is called a lost tick.

If the virtual machine is running too slowly, perhaps as a result of competition for CPU time from other virtual machines or processes running on the host machine, it may be impossible to feed the virtual machine enough interrupts to keep up with real time. In current VMware products, if the backlog of interrupts grows beyond 60 seconds, the virtual machine gives up on catching up, simply setting its record of the backlog to zero. After this happens, if VMware Tools is installed in the guest operating system and its clock synchronization feature is enabled, VMware Tools corrects the clock reading in the guest operating system sometime within the next minute by synchronizing the guest operating system time to match the host machine's clock. The virtual machine then resumes keeping track of its backlog and catching up any new backlog that accumulates.

Another problem with timer interrupts is that they cause a scalability issue as more and more virtual machines are run on the same physical machine. Even when a virtual machine is otherwise completely idle, it must run briefly each time it receives a timer interrupt. If a virtual machine is requesting 100 interrupts per second, it thus becomes ready to run at least 100 times per second, at evenly spaced intervals. So roughly speaking, if N virtual machines are running, processing the interrupts imposes a background load of $100 \times N$ context switches per second (even if all the virtual machines are idle). Virtual machines that request 1,000 interrupts per second create 10 times the context switching load, and so forth.

Tickless Timekeeping

A growing number of PC-based operating systems use tickless timekeeping. This form of timekeeping is relatively easy to support in a virtual machine and has several advantages, but there are still a few challenges.

On the positive side, when the guest operating system is not counting timer interrupts for timekeeping purposes, there is no need for the virtual machine to keep track of an interrupt backlog and catch up if the number of interrupts delivered has fallen behind real time. Late interrupts can simply be allowed to pile up and merge together, without concern for clock slippage caused by lost ticks. This saves CPU time that would otherwise be consumed in handling the late interrupts. Further, the guest operating system's view of time is more accurate, because its clock does not fall behind real time while the virtual machine is not running or is running slowly.

In order to achieve these advantages, however, the virtual machine needs to know that the guest operating system is using tickless timekeeping. The virtual machine must assume tick counting in the absence of knowledge to the contrary, because if the guest operating system is in fact counting timer interrupts, it is incorrect to drop any. VMware products use multiple methods to detect tickless timekeeping. First, if the guest has not programmed any of the virtual timer devices to generate periodic interrupts, it is safe to assume that tick counting is not in use. However, some operating systems do program one or more timer devices for periodic interrupts even when using tickless timekeeping. In such cases the use of tickless timekeeping can usually be inferred from the guest operating system type. Alternatively, software in the virtual machine can make a hypercall to inform the virtual machine that it is tickless.

An additional challenge for both forms of timekeeping is that virtual machines occasionally run highly time-sensitive code—for example, measuring the number of iterations of a specific loop that can run in a given amount of real time. In some cases, such code may behave better under the tick-counting style of timekeeping, in which the guest operating system's view of time appears to slow down or stop while the virtual machine is not running.

Initializing and Correcting Wall-Clock Time

A guest operating system faces the same basic challenges in keeping accurate wall-clock time when running in either a virtual or physical machine: initializing the clock to the correct time when booting, and updating the clock accurately as time passes.

For initializing the clock, a VMware virtual machine provides mechanisms similar to those of a physical machine: a virtual battery-backed CMOS clock and virtual network cards that can be used to fetch the time from a network time server. One further mechanism is also provided: VMware Tools resets the guest operating system's clock to match the host's clock upon startup. The interface between guest and host uses UTC (Coordinated Universal Time, also known as Greenwich Mean Time or GMT), so the guest and host do not have to be in the same time zone.

Virtual machines also have a further issue: when the virtual machine is resumed from suspend or restored from a snapshot, the guest operating system's wall-clock time remains at the value it had at the time of the suspension or snapshot and must be updated. VMware Tools handles this issue too, setting the virtual machine's clock to match the host's clock upon resume or restore. However, because users sometimes need a virtual machine to have its clock set to a fictitious time unrelated to the time kept on the host, VMware Tools can optionally be told never to change the virtual machine's clock.

Updating the clock accurately over the long term is challenging because the timer devices in physical machines tend to drift, typically running as much as 100 parts per million fast or slow, with the rate varying with temperature. The virtual timer devices in a virtual machine have the same amount of inherent drift as the underlying hardware on the host, and additional drift and inaccuracy can arise as a result of such factors as round-off error and lost ticks. In a physical machine, it is generally necessary to run network clock synchronization software such as NTP or the Windows Time Service to keep time accurately over the long term. The same applies to virtual machines, and the same clock synchronization software can be used, although it sometimes needs to be configured specially in order to deal with the less smooth behavior of virtual timer devices. VMware Tools can also optionally be used to correct long-term drift and errors by periodically resynchronizing the virtual machine's clock to the host's clock, but the current version at this writing is limited. In particular, in guest operating systems other than NetWare, it does not correct errors in which the guest clock is ahead of real time, only those in which the guest clock is behind.

Time and Frequency Units

The following table provides a quick summary of units in which time and frequency are measured:

Table 1. Units for Measuring Time and Frequency

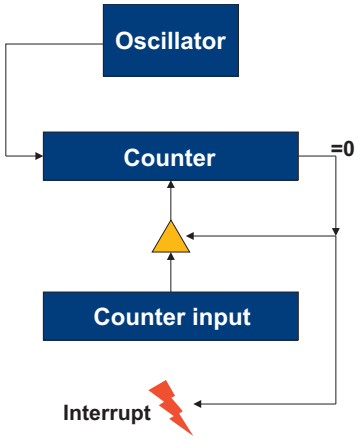
Abbreviation	Description
s	Seconds
ms	Milliseconds (1/1000 second)
µs	Microseconds (10^{-6} seconds)
ns	Nanoseconds (10^{-9} seconds)
Hz	Frequency (cycles or other events per second)
KHz	Kilohertz (1000 cycles or events per second)
MHz	Megahertz (10^{6} cycles or events per second)
GHz	Gigahertz (10^{9} cycles or events per second)

PC Timer Hardware

For historical reasons, PCs contain several different devices that can be used to keep track of time. Different guest operating systems make different choices about which of these devices to use and how to use them. Using several of the devices in combination is important in many guest operating systems. Sometimes, one device that runs at a known speed is used to measure the speed of another device. Sometimes a fine-grained timing device is used to add additional precision to the tick count obtained from a more coarse-grained timing device. Thus, it is necessary to support all these devices in a virtual machine, and the times read from different devices usually must appear to be consistent with one another, even when they are somewhat inconsistent with real time.

All PC timer devices can be described using roughly the same block diagram, as shown in Figure 1. Not all the devices have all the features shown, and some have additional features, but the diagram is a useful abstraction.

Figure 1. Abstract Timer Device



The oscillator provides a fixed input frequency to the timer device. The frequency may be specified, or the operating system may have to measure it at startup time. The counter may be readable or writable by software. The counter counts down one unit for each cycle of the oscillator. When the counter reaches zero, it generates an output signal that may interrupt the processor. At this point, if the timer is set to one-shot mode, it stops; if set to periodic mode, it continues counting. There may also be a counter input register whose value is loaded into the counter when it reaches zero; this register allows software to control the timer period. Some real timer devices count up instead of down and have a register whose value is compared with the counter to determine when to interrupt and restart the count at zero, but both count-up and count-down timer designs provide equivalent functionality.

Common PC timer devices include the programmable interval timer (PIT), the CMOS real time clock (RTC), the local advanced programmable interrupt controller (APIC) timers, the advanced configuration and power interface (ACPI) timer, the time stamp counter (TSC), and the high precision event timer (HPET).

PIT

The PIT is the oldest PC timer device. It uses a crystal-controlled 1.193182MHz input oscillator and has 16-bit counter and counter input registers. The oscillator frequency was not chosen for convenient timekeeping; it was simply a handy frequency available when the first PC was designed. (The oscillator frequency is one-third of the standard NTSC television color burst frequency.) The PIT device actually contains three identical timers that are connected in different ways to the rest of the computer. Timer 0 can generate an interrupt and is suitable for system timekeeping. Timer 1 was historically used for RAM refresh and is typically programmed for a 15 microsecond period by the PC BIOS. Timer 2 is wired to the PC speaker for tone generation.

CMOS RTC

The CMOS RTC is part of the battery-backed memory device that keeps a PC's BIOS settings stable while the PC is powered off. The name CMOS comes from the low-power integrated circuit technology in which this device was originally implemented. There are two main time-related features in the RTC. First, there is a continuously running time of day (TOD) clock that keeps time in year/month/day hour:minute:second format. This clock can be read only to the nearest second. There is also a timer that can generate periodic interrupts at any power-of-two rate from 2Hz to 8192Hz. This timer fits the block diagram model in Figure 1, with the restriction that the counter cannot be read or written, and the counter input can be set only to a power of two.

Two other interrupts can also be enabled: the update interrupt and the alarm interrupt. The update interrupt occurs once per second. It is supposed to reflect the TOD clock turning over to the next second. The alarm interrupt occurs when the time of day matches a specified value or pattern.

Local APIC Timer

The local APIC is a part of the interrupt routing logic in modern PCs. In a multiprocessor system, there is one local APIC per processor. On current processors, the local APIC is integrated onto the processor chip. The local APIC includes a timer device with 32-bit counter and counter input registers. The input frequency is typically the processor's base front-side memory bus frequency (before the multiplication by two or four for DDR or quad-pumped memory). Thus, this timer is much finer-grained and has a wider counter than the PIT or CMOS timers, but software does not have a reliable way to determine its frequency. Generally, the only way to determine the local APIC timer's frequency is to measure it using the PIT or CMOS timer, which yields only an approximate result.

ACPI Timer

The ACPI timer is an additional system timer that is required as part of the ACPI specification. This timer is also known as the power management (PM) timer or the chipset timer. It has a 24-bit counter that increments at 3.579545MHz (three times the PIT frequency). The timer can be programmed to generate an interrupt when its high-order bit changes value. There is no counter input register; the counter always rolls over. (That is, when the counter reaches the maximum 24-bit binary value, it goes back to zero and continues counting from there.) The ACPI timer continues running in some power saving modes in which other timers are stopped or slowed. The ACPI timer is relatively slow to read (typically 1–2μs).

TSC

The TSC is a 64-bit cycle counter on Pentium CPUs and newer processors. The TSC runs off the CPU clock oscillator, typically 2GHz or more on current systems. At current processor speeds it would take years to roll over. The TSC cannot generate interrupts and has no counter input register. The TSC can be read by software in one instruction (`rdtsc`). The `rdtsc` instruction is normally available in user mode, but operating system software can choose to make it unavailable. The TSC is, by far, the finest grained, widest, and most convenient timer device to access. However, the TSC also has several drawbacks:

- As with the local APIC timer, software does not have a reliable way to determine the TSC's input frequency. Generally, the only way to determine the TSC's frequency is to measure it approximately using the PIT or CMOS timer.
- Several forms of power management technology vary the processor's clock speed dynamically and thus change the TSC's input oscillator rate with little or no notice. In addition, AMD Opteron K8 processors drop some cycles from the TSC when entering and leaving a halt state if the halt clock ramping feature is enabled, even though the TSC rate does not change. The latest processors from Intel and AMD no longer have these limitations, however.
- Some processors stop the TSC in their lower-power halt states (the ACPI C3 state and below).
- On shared-bus SMP machines, all the TSCs run off a common clock oscillator, so (in the absence of the issues noted above) they can be synchronized closely with each other at startup time and thereafter treated essentially as a single system-wide clock. This does not work on IBM x-Series NUMA (non-uniform memory access) machines and their derivatives, however. In these machines, different NUMA nodes run off separate clock oscillators. Although the nominal frequencies of the oscillators on each NUMA node are the same, each oscillator is controlled by a separate crystal with its own distinct drift from the nominal frequency. In addition, the clock rates are intentionally varied dynamically over a small range (2 percent or so) to reduce the effects of emitted RF (radio frequency) noise, a technique called spread-spectrum clocking, and this variation is not in step across different nodes.

Despite these drawbacks of the TSC, both operating systems and application programs frequently use the TSC for timekeeping.

HPET

The HPET is a device available in some newer PCs. Many PC systems do not have this device and operating systems generally do not require it, though some can use it if available. The HPET has one central up-counter that runs continuously unless stopped by software. It may be 32 or 64 bits wide. The counter's period can be read from a register. The HPET provides multiple timers, each consisting of a timeout register that is compared with the central counter. When a timeout value matches, the corresponding timer fires. If the timer is set to be periodic, the HPET hardware automatically adds its period to the compare register, thus computing the next time for this timer to fire.

The HPET has a few drawbacks. The specification does not require the timer to be particularly fine-grained, to have low drift, or to be fast to read. Some typical implementations run the counter at about 18MHz and require about the same amount of time (1–2μs) to read the HPET as with the ACPI timer. Implementations have been observed in which the period register is off by 800 parts per million or more. A drawback of the general design is that setting a timeout races with the counter itself. If software tries to set a short timeout, but for any reason its write to the HPET is delayed beyond the point at which the timeout is to expire, the timeout is effectively set far in the future instead (about 2^{32} or 2^{64} counts). Software can stop the central counter, but doing so would spoil its usefulness for long-term timekeeping.

The HPET is designed to be able to replace the PIT and CMOS periodic timers by driving the interrupt lines to which the PIT and CMOS timers are normally connected. Most current hardware platforms still have physical PIT and CMOS timers and do not need to use the HPET to replace them.

VMware Timer Virtualization

VMware products use a patent-pending technique to allow the many timer devices in a virtual machine to fall behind real time and catch up as needed, yet remain sufficiently consistent with one another that software running in the virtual machine is not disrupted by anomalous time readings. In VMware terminology, the time that is visible to virtual machines on their timer devices is called apparent time. Generally, the timer devices in a virtual machine operate identically to the corresponding timer devices in a physical machine, but they show apparent time instead of real time. The following sections note some exceptions to this rule and provide some additional details about each emulated timer device.

Virtual PIT

VMware products fully emulate the timing functions of all three timers in the PIT device. In addition, when the guest operating system programs the speaker timer to generate a sound, the virtual machine requests a beep sound from the host machine. However, the sound generated on the host may not be the requested frequency or duration.

Virtual CMOS RTC

Current VMware products emulate all the timing functions of the CMOS RTC, including the time of day clock and the periodic, update, and alarm interrupts that the CMOS RTC provides.

Many guest operating systems use the CMOS periodic interrupt as the main system timer, so VMware products run it in apparent time to be consistent with the other timer devices. Some guest operating systems use the CMOS update interrupt to count off precisely one second to measure the CPU speed or the speed of other timer devices, so VMware products run the CMOS update interrupt in apparent time as well.

In contrast, VMware products base the virtual CMOS TOD clock directly on the real time as known to the host system, not on apparent time. This choice makes sense because guest operating systems generally read the CMOS TOD clock only to initialize the system time at power on and occasionally to check the system time for correctness. Operating systems use the CMOS TOD clock this way because it provides time only to the nearest second but is battery backed and thus continues to keep time even when the system loses power or is restarted.

Specifically, the CMOS TOD clock shows UTC as kept by the host operating system software, plus an offset. The offset from UTC is stored in the virtual machine's `nvram` file along with the rest of the contents of the virtual machine's CMOS nonvolatile memory. The offset is needed because many guest operating systems want the CMOS TOD clock to show the time in the current local time zone, not in UTC. When you create a new virtual machine (or delete the `nvram` file of an existing virtual machine) and power it on, the offset is initialized, by default, to the difference of the host operating system's local time zone from UTC. If software running in the virtual machine writes a new time to the CMOS TOD clock, the offset is updated.

You can force the CMOS TOD clock's offset to be initialized to a specific value at power on. To do so, set the option `rtc.diffFromUTC` in the virtual machine's `.vmx` configuration file to a value in seconds. For example, setting `rtc.diffFromUTC = 0` sets the clock to UTC at power on, while setting `rtc.diffFromUTC = –25200` sets it to Pacific Daylight Time, seven hours earlier than UTC. The guest operating system can still change the offset value after power on by writing a new time to the CMOS TOD clock.

You can also force the CMOS TOD clock to start at a specified time whenever the virtual machine is powered on, independent of the real time. To do this, set the configuration file option `rtc.startTime`. The value you specify is in seconds since Jan 1, 1970 00:00 UTC, but it is converted to the local time zone of the host operating system before setting the CMOS TOD clock (under the assumption that the guest operating system wants the CMOS TOD clock to read in local time). If your guest operating system is running the CMOS TOD clock in UTC or some other time zone, you should correct for this when setting `rtc.startTime`.

The virtual CMOS TOD clock has the following limitation: Because the clock is implemented as an offset from the host operating system's software clock, it changes value if you change the host operating system time. (Changing the host time zone has no effect, only changing the actual time.) In most cases this effect is harmless, but it does mean that you should never use a virtual machine as a time server providing time to the host operating system that it is running on. Doing this can create a harmful positive feedback loop, in which any change made to the host time incorrectly changes the guest time, too, causing the host time to appear wrong again, which causes a further change to the host time, etc. Whether this effect occurs and how severe it is depends on how the guest operating system uses the CMOS TOD clock. Some guest operating systems may not use the CMOS TOD clock at all, in which case the problem does not occur. Some guests synchronize to the CMOS TOD clock only at boot time, in which case the problem does occur but the system goes around its feedback loop only once per guest boot. You can use `rtc.diffFromUTC` to break such a feedback loop, but it is better to avoid the loop in the first place by not using the virtual machine as a time server for the host. Some guest operating systems periodically resynchronize to the CMOS TOD clock (say, once per hour), in which case the feedback is more rapid and `rtc.diffFromUTC` cannot break the loop.

Because the alarm interrupt is designed to be triggered when the CMOS TOD clock reaches a specific value, the alarm interrupt also operates in real time, not apparent time.

The choice of real or apparent time for each feature of the CMOS RTC device reflects the way guest operating systems commonly use the device. Guest operating systems typically have no difficulty with part of the device operating in apparent time and other parts operating in real time. However, one unsupported guest operating system (USL Unix System V Release 4.21) is known to crash if it sees the CMOS device's update-in-progress (UIP) bit set while starting up. It is not known whether this crash would occur on real hardware or whether the guest operating system is confused by the fact that the update interrupt, the UIP bit, and the rollover of the CMOS TOD clock to the next second do not all occur at the same moment, as they would on real hardware. You can work around this problem by setting `rtc.doUIP = FALSE` in the virtual machine's configuration file, which forces the UIP bit to always return 0.

NOTE Do not use the `rtc.doUIP = FALSE` setting unless you are running a guest operating system that requires it. Setting this value for other guest operating systems may prevent timekeeping from working correctly.

Virtual Local APIC Timer

VMware products fully emulate the local APIC timer on each virtual CPU. The timer's frequency is not dependent on the host's APIC timer frequency.

In most cases, the local APIC timer runs in apparent time, matching the other timer devices. However, some VMware products are able to recognize cases in which the guest operating system is using tickless timekeeping but has nevertheless set up a periodic local APIC timer interrupt. In these cases, the local APIC timer runs in a "lazy" mode, in which reading its counter returns the current apparent time, but late APIC timer interrupts are allowed to pile up and merge rather than being accumulated as a backlog and causing apparent time to be held back until the backlog is caught up. Also, APIC timer interrupts on different virtual CPUs are allowed to occur slightly out of order.

Virtual ACPI Timer

VMware products fully emulate a 24-bit ACPI timer. The timer runs in apparent time, matching the other timer devices. It generates an interrupt when the high-order bit changes value.

Virtual TSC

Current VMware products virtualize the TSC in apparent time. The virtual TSC stays in step with the other timer devices visible in the virtual machine. Like those devices, the virtual TSC falls behind real time when there is a backlog of timer interrupts and catches up as the backlog is cleared. Thus, the virtual TSC does not count cycles of code run on the virtual CPU; it advances even when the virtual CPU is not running. The virtual TSC also does not match the TSC value on the host hardware. When a virtual machine is powered on, its virtual TSC is set, by default, to run at the same rate as the host TSC. If the virtual machine is then moved to a different host without being powered off (that is, either using VMotion or suspending the virtual machine on one host and resuming it on another), the virtual TSC continues to run at its original power-on rate, not at the host TSC rate on the new host machine.

You can force the virtual TSC's rate to a specific value *N* (in cycles per second or Hz) by adding the setting `timeTracker.apparentHz = N` to the virtual machine's `.vmx` configuration file. This feature is rarely needed. One possible use is to test for bugs in guest operating systems—for example, Linux 2.2 kernels hang during startup if the TSC runs faster than 4GHz. Note that this feature does not change the rate at which instructions are executed. In particular, you cannot make programs run more slowly by setting the virtual TSC's rate to a lower value.

You can disable virtualization of the TSC by adding the setting `monitor_control.virtual_rdtsc = FALSE` to the virtual machine's `.vmx` configuration file. This feature is no longer recommended for use. When you disable virtualization of the TSC, reading the TSC from within the virtual machine returns the physical machine's TSC value, and writing the TSC from within the virtual machine has no effect. Migrating the virtual machine to another host, resuming it from suspended state, or reverting to a snapshot causes the TSC to jump discontinuously. Some guest operating systems fail to boot or exhibit other timekeeping problems when TSC virtualization is disabled. In the past, this feature has sometimes been recommended to improve performance

of applications that read the TSC frequently, but performance of the virtual TSC has been improved substantially in current products. The feature has also been recommended for use when performing measurements that require a precise source of real time in the virtual machine, but for this purpose, the pseudoperformance counters discussed in the next section are a better choice.

Pseudoperformance Counters

For certain applications it can be useful to have direct access to real time (as opposed to apparent time) within a virtual machine. For example, you may be writing performance measuring software that is aware it is running in a virtual machine and does not require its fine-grained timer to stay in step with the number of interrupts delivered on other timer devices.

VMware virtual machines provide a set of pseudoperformance counters that software running in the virtual machine can read with the `rdpmc` instruction to obtain fine-grained time. To enable this feature, use the following configuration file setting:

```
monitor_control.pseudo_perfctr = TRUE
```

The following machine instructions then become available:

Table 2. Instructions Available When Pseudoperformance Counters Are Enabled

Instruction	Time Value Returned
`rdpmc 0x10000`	Physical host TSC
`rdpmc 0x10001`	Elapsed real time in ns
`rdpmc 0x10002`	Elapsed apparent time in ns

Although the `rdpmc` instruction normally is privileged unless the PCE flag is set in the CR4 control register, a VMware virtual machine permits the above pseudoperformance counters to be read from user space regardless of the setting of the PCE flag. Note that the pseudoperformance counter feature uses a trap to catch a privileged machine instruction issued by software running in the virtual machine and thus has more overhead than reading a performance counter or the TSC on physical hardware.

There are some limitations. Some or all of these counters may not be available on older versions of VMware products. In particular, elapsed real time and elapsed apparent time were first introduced in VMware ESX 3.5 and VMware Workstation 6.5. The zero point for the counters is currently unspecified. The physical host TSC may change its counting rate, jump to a different value, or both when the virtual machine migrates to a different host or is resumed from suspend or reverted to a snapshot. The elapsed real time counter runs at a constant rate but may jump to a different value when the virtual machine migrates to a different host or is resumed from suspend or reverted to a snapshot.

Virtual HPET

Current VMware products do not provide a virtual HPET, because currently supported guest operating systems do not require one.

Other Time-Dependent Devices

Computer generation of sound is time-sensitive. The sounds that a virtual machine generates are always played by the host machine's sound card at the correct sample rate, regardless of timer behavior in the virtual machine, so they always play at the proper pitch. Also, there is enough buffering between the virtual sound card of the virtual machine and the host machine's sound card so that sounds usually play continuously. However, there can be gaps or stuttering if the virtual machine falls far enough behind that the supply of buffered sound information available to play is exhausted.

Playback of MIDI music (as well as some other forms of multimedia), however, requires software to provide delays for the correct amount of time between notes or other events. Thus, playback can slow down or speed up if the apparent time deviates too far from real time.

VGA video cards produce vertical and horizontal blanking signals that depend on a monitor's video scan rate. VMware virtual machines currently make no attempt to emulate these signals with accurate timing. There is very little software that uses these signals for timing, but a few old games do use them. These games currently are not playable in a virtual machine.

VMI Paravirtual Timer

The Virtual Machine Interface (VMI) is an open paravirtualization interface developed by VMware with input from the Linux community. VMI is an open standard, the specification for which is available at http://www.vmware.com/pdf/vmi_specs.pdf. VMI is currently defined only for 32-bit guests. VMware products beginning with Workstation 6 and ESX 3.5 support VMI.

VMI includes a paravirtual timer device that the guest operating system kernel can use for tickless timekeeping. In addition, VMI allows the guest kernel to explicitly account for "stolen time"; that is, time when the guest operating system was ready to run but the virtual machine was descheduled by the host scheduler.

VMI could be used by any guest operating system, but currently only Linux uses it. See "Linux" on page 11.

Timekeeping in Specific Operating Systems

This section details some of the peculiarities of specific operating systems that affect their timekeeping performance when they are run as guests in virtual machines. A few of these issues also affect timekeeping behavior when these operating systems are run as hosts for VMware Workstation and other VMware hosted products.

Microsoft Windows

Microsoft Windows operating systems generally keep time by counting timer interrupts (ticks). System time of day is precise only to the nearest tick. The timer device used and the number of interrupts generated per second vary depending on which specific version of Microsoft Windows and which Windows hardware abstraction layer (HAL) are installed. Some uniprocessor Windows configurations use the PIT as their main system timer, but multiprocessor HALs and some ACPI uniprocessor HALs use the CMOS periodic timer instead. For systems using the PIT, the base interrupt rate is usually 100Hz, although Windows 98 uses 200Hz. For systems that use the CMOS timer, the base interrupt rate is usually 64Hz.

Microsoft Windows also has a feature called the multimedia timer API that can raise the timer rate to as high as 1024Hz (or 1000Hz on systems that use the PIT) when it is used. For example, if your virtual machine has the Apple QuickTime icon in the system tray, even if QuickTime is not playing a movie, the guest operating system timer rate is raised to 1024Hz. This feature is not used exclusively by multimedia applications. For example, some implementations of the Java runtime environment raise the timer rate to 1024Hz, so running any Java application may raise your timer rate, depending on the version of the runtime you are using. This feature is also used by VMware hosted products running on a Windows host system, to handle cases in which one or more of the currently running virtual machines require a higher virtual timer interrupt rate than the host's default physical interrupt rate.

Microsoft Windows has an additional time measurement feature accessed through the `QueryPerformanceCounter` system call. This name is a misnomer, because the call never accesses the CPU's performance counter registers. Instead, it reads one of the timer devices that have a counter, allowing time measurement with a finer granularity than the interrupt-counting system time of day clock. Which timer device is used (the ACPI timer, the TSC, the PIT, or some other device) depends on the specific Windows version and HAL in use.

Some versions of Windows, especially multiprocessor versions, set the TSC register to zero during their startup sequence, in part to ensure that the TSCs of all the processors are synchronized. Microsoft Windows also measures the speed of each processor by comparing the TSC against one of the other system timers during startup, and this code also sets the TSC to zero in some cases.

Some multiprocessor versions of the Windows operating system program the local APIC timers to generate one interrupt per second. Other versions of Windows do not use these timers at all.

Some multiprocessor versions of Windows route the main system timer interrupt as a broadcast to all processors. Others route this interrupt only to the primary processor and use interprocessor interrupts for scheduler time slicing on secondary processors.

To initialize the system time of day on startup, Microsoft Windows reads the battery-backed CMOS TOD clock. Occasionally, Windows also writes to this clock so that the time is approximately correct on the next startup. Windows keeps the CMOS TOD clock in local time, so in regions that turn their clocks ahead by an hour during the summer, Windows must update the CMOS TOD clock twice a year to reflect the change. Some rare failure modes can put the CMOS TOD clock out of step with the Windows registry setting that records whether it has been updated, causing the Windows clock to be off by an hour after the next reboot. If VMware Tools is installed in the virtual machine, it corrects any such error at boot time.

A daemon present in Windows NT-family systems (that is, Windows NT 4.0 and later) checks the system time of day against the CMOS TOD clock once per hour. If the system time is off by more than 60 seconds, it is reset to match the TOD clock. This behavior is generally harmless in a virtual machine and may be useful in some cases, such as when VMware Tools or other synchronization software is not in use. One possible (though rare) problem can occur if the daemon sets the clock ahead while the virtual machine is in the process of catching up on an interrupt backlog. Because the virtual machine is not aware that the guest operating system clock has been reset, it continues catching up, causing the clock to overshoot real time. If you turn on periodic clock synchronization in VMware Tools, it disables this daemon.

For a discussion of `W32Time` and other Windows clock synchronization software, see "Synchronizing Virtual Machines and Hosts with Real Time" on page 14.

Linux

Timekeeping in Linux has changed a great deal over its history. Recently, the direction of kernel development has been toward better behavior in a virtual machine. However, along the way, a number of kernels have had specific bugs that are strongly exposed when run in a virtual machine. Some kernels have very high interrupt rates, resulting in poor timekeeping performance and imposing excessive host load even when the virtual machine is idle. In most cases, the latest VMware-supported version of a Linux distribution has the best timekeeping behavior. See VMware knowledge base article 1006427 (http://kb.vmware.com/kb/1006427) for specific recommendations, including workarounds for bugs and performance issues with specific distribution vendor kernels. The remainder of this section describes the overall development and characteristics of the Linux timekeeping implementation in more detail.

Linux kernel version 2.4 and earlier versions of 2.6 used tick counting exclusively, with PIT 0 usually used as the source of ticks. More recently, a series of changes amounting to a rewrite of the timekeeping subsystem were made to the kernel. The first major round of changes, which went into the 32-bit kernel in version 2.6.18 and the 64-bit kernel in 2.6.21, added an abstraction layer called clocksource to the timekeeping subsystem. The kernel can select from several clock sources at boot time. Generally, each one uses a different hardware timer device with appropriate support code to implement the clocksource interface. Almost all the provided clock sources are tickless, including all the commonly used ones. The next major round of changes, completed in 32-bit 2.6.21 and 64-bit 2.6.24, add the clockevents abstraction to the kernel. These changes add the `NO_HZ` kernel configuration option, which, when enabled at kernel compile time, switches the kernel to using an aperiodic (one-shot) interrupt for system timer callbacks, process accounting, and scheduling.

Versions of Linux prior to the introduction of `NO_HZ` require a periodic interrupt for scheduler time slicing and statistical process accounting. Most configurations use the local APIC timer on each CPU to generate scheduler interrupts for that CPU, but in some uniprocessor configurations, the scheduler is driven from the same interrupt used for timekeeping.

User applications on Linux can request additional timer interrupts using the `/dev/rtc` device. These interrupts come either from the CMOS periodic timer or the HPET. This feature is used by some multimedia software. It is also used by VMware hosted products running on a Linux host system, to handle cases in which one or more of the currently running virtual machines requires a higher virtual timer interrupt rate than the host's default physical interrupt rate. See VMware Knowledge Base article 892 (http://kb.vmware.com/kb/892). The Linux implementation of this feature using the HPET can sometimes stop delivering interrupts because of the timeout-setting race mentioned in "HPET" on page 6.

Most Linux distributions are set up to initialize the system time from the battery-backed CMOS TOD clock at startup and to write the system time back to the CMOS TOD clock at shutdown. In some cases, Linux kernels also write the system time to the CMOS TOD clock periodically (once every 11 minutes). You can manually read or set the CMOS TOD clock using the `/sbin/hwclock` program.

Kernels Before Clocksource

Linux kernels prior to the introduction of the clocksource abstraction count periodic timer interrupts as their basic method of timekeeping. Linux kernels generally use PIT 0 as their main source of timer interrupts. The interrupt rate used depends on the kernel version. Linux 2.4 and earlier kernels generally program the PIT 0 timer to deliver interrupts at 100Hz. Some vendor patches to 2.4 kernels increase this rate. In particular, the initial release of Red Hat Linux 8 and some updates to Red Hat Linux 7 used 512Hz, but later updates reverted to the standard 100Hz rate. SUSE Linux Professional 9.0 uses 1000Hz when the desktop boot-time option is provided to the kernel, and the SUSE installation program sets this option by default. Early Linux 2.6 kernels used a rate of 1000Hz. This rate was later made configurable at kernel compile time, with 100Hz, 250Hz, and 1000Hz as standard choices and 250Hz as the default; however, some vendors (including Red Hat in the Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5 series) continued to ship kernels configured for 1000Hz. The latest versions in both the Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5 series include a `divider=` boot-time option that can reduce the interrupt rate—for example, `divider=10` reduces the interrupt rate to 100Hz.

As mentioned above, most kernels also program the local APIC timer on each CPU to deliver periodic interrupts to drive the scheduler. These interrupts occur at approximately the same base rate as the PIT 0 timer. Thus, a one-CPU virtual machine running an SMP Linux 2.4 kernel requires a total of 200 timer interrupts per second across all sources, while a two-CPU virtual machine requires 300 interrupts per second. A one-CPU Linux 2.6 kernel virtual machine that uses tick counting for timekeeping and the local APIC timer for scheduling requires a total of 2000 timer interrupts per second, while a two-CPU virtual machine requires 3000 interrupts per second.

32-bit Linux kernel 2.4 and earlier versions interpolate the system time (as returned by the `gettimeofday` system call) between timer interrupts using an algorithm that is somewhat prone to errors. First, the kernel counts PIT timer interrupts to keep track of time to the nearest 10 milliseconds. When a timer interrupt is received, the kernel reads the PIT counter to measure and correct for the latency in handling the interrupt. The kernel also reads and records the TSC at this point. On each call to `gettimeofday`, the kernel reads the TSC again and adds the change since the last timer interrupt was processed to compute the current time. Implementations of this algorithm have had various problems that result in incorrect time readings being produced when certain race conditions occur. These problems are fairly rare on real hardware but are more frequent in a virtual machine. The algorithm is also sensitive to lost ticks (as described earlier), and these seem to occur more often in a virtual machine than on real hardware. As a result, if you run a program that loops calling `gettimeofday` repeatedly, you may occasionally see the value go backward. This occurs both on real hardware and in a virtual machine but is more frequent in a virtual machine.

Most 32-bit versions of Linux kernel 2.6 that predate clocksource implement several different algorithms for interpolating the system time and let you choose among them with the `clock=` kernel command line option. Unfortunately, all the available options have some drawbacks.

Two of the algorithms incorporate code that attempts to detect lost ticks from nonprocessed timer interrupts automatically and add extra ticks to correct for the time loss. Unfortunately, these algorithms often overcorrect: they add extra, spurious ticks to the operating system clock when timer interrupt handling is delayed such that two interrupts are handled in close succession but neither is lost. Such bunching of interrupts occurs occasionally on real hardware, usually because a CPU is busy handling other tasks while interrupts are temporarily disabled. This problem occurs much more frequently in a virtual machine because of the virtual machine's need to share the real CPU with other processes. Thus, this problem can cause the clock to run too fast both on real hardware and in a virtual machine, but the effect is much more noticeable in a virtual machine. The following paragraphs describe the time interpolation algorithms of 32-bit Linux 2.6 kernels predating clocksource that are usable in a virtual machine:

- The option `clock=tsc` selects an algorithm that makes use of the PIT counter and the TSC for time interpolation. This algorithm is similar to that of Linux kernel 2.4 but incorporates lost tick correction. As previously noted, the methods used to adjust time for lost ticks may overcorrect, making the clock run too fast. Time gains of 10 percent or more have been observed when running this algorithm in a virtual machine. This option is the default on some kernels.
- The option `clock=pmtmr` selects a simpler but more robust algorithm that makes use of the ACPI timer for interpolation. This option also includes lost tick correction code that may cause time gains. However, when used in a virtual machine, time gains from using this option are much smaller. This option is usable in a virtual machine, if you can tolerate the small time gain. This option is the default on some kernels.
- The option `clock=pit` uses only the PIT counter for interpolation. It does not include lost tick correction code, so it does not gain time, but it does lose time when a tick is actually lost. Unfortunately, this option has a different bug on most kernels, such that if you write a program that calls `gettimeofday` repeatedly, it frequently jumps backward by about 1ms, then corrects itself.

The 64-bit Linux kernels that predate clocksource implement timekeeping with a method different from that used in the corresponding 32-bit kernels. Unfortunately, this implementation also has problems with lost tick correction code that overcorrects and causes a time gain. Even some 64-bit kernels in the 2.4 series have lost tick overcorrection—for example, the kernel included in 64-bit Red Hat Enterprise Linux 3 has this problem. The 64-bit timekeeping implementation has a slightly different set of available algorithms and is controlled by a different boot-time option.

- One algorithm is similar to the 32-bit `clock=tsc` algorithm discussed above and has the same problem of severe lost tick overcorrection. This option is the default on most kernels and is the only one available on older kernels.
- An alternative algorithm is similar to the 32-bit `clock=pmtmr` algorithm discussed above. It has the same problem of minor lost tick overcorrection. On some kernels this algorithm is not available at all; on some it is selected automatically for AMD processors but cannot be selected manually. On most kernels, however, this algorithm can be selected using the `notsc` kernel command line option.

See VMware knowledge base article 1006427 (http://kb.vmware.com/kb/1006427) for recommended options to use on specific vendor kernels.

Clocksource Kernels

With the new clocksource abstraction, the kernel's high-level timekeeping code basically deals only with wall-clock time and NTP rate correction. It calls into a lower-level clocksource driver to read a counter that reflects the raw amount of time (without rate correction) that has passed since boot. The available clocksource drivers generally do not use any of the problematic techniques from earlier Linux timekeeping implementations, such as using one timer device to interpolate between the ticks of another or doing lost tick compensation. In fact, most of the clocksource drivers are tickless. The TSC clocksource (usually the default) basically just reads the TSC value and returns it. The ACPI PM timer clocksource is similar, as the kernel handles timers that wrap (which occurs about every four seconds with a 24-bit ACPI PM timer) and extends their range automatically.

The clocksource abstraction is a good match for virtual machines, though not perfect. The TSC does not run at a precisely specified rate, so the guest operating system has to measure its rate at boot time, and this measurement is always somewhat inaccurate. Running NTP or other clock synchronization software in the guest can compensate for this issue, however. The ACPI PM timer does run at a precisely specified rate but is slower to read than the TSC. Also, when clocksource is used without `NO_HZ`, the guest operating system still programs a timer to interrupt periodically, so by default, the virtual machine still keeps track of a backlog of timer interrupts and tries to catch up gradually.

The `NO_HZ` option provides a further significant improvement. Because the guest operating system does not schedule any periodic timers, the virtual machine can never have a backlog greater than one timer interrupt, so apparent time does not fall far behind real time and catches up very quickly. Also important, `NO_HZ` tends to reduce the overall average rate of virtual timer interrupts, improving system throughput and scalability to larger numbers of virtual machines per host.

Alternatively, even on kernels without NO_HZ, software running in the virtual machine can make a hypercall to inform the virtual machine that it is tickless. For example, the 64-bit SUSE Linux Enterprise Server 10 SP2 kernel does this.

Paravirtual Kernels

With some 32-bit kernel versions, you can use VMI (see "VMI Paravirtual Timer" on page 10) to obtain tickless timekeeping in a virtual machine. The VMI patches developed for kernel 2.6.20 and earlier include changes to the timekeeping subsystem that use the VMI paravirtual timer device for tickless timekeeping and stolen-time accounting. Some distribution vendors have shipped kernels that include this version of VMI, so if you run one of these distributions in a virtual machine, you get the benefits of these changes. In particular, both Ubuntu 7.04 (2.6.20) and SUSE Linux Enterprise Server 10 SP2 (2.6.16) ship with VMI enabled in the default kernel.

The VMI patches were accepted into the mainline 32-bit kernel in version 2.6.21. However, the clocksource abstraction was also added to the kernel in this version, making tickless timekeeping available using the generic high-level timekeeping code. Accordingly, the timekeeping portion of the VMI patches was dropped at this point, because it was no longer needed. (Stolen time accounting was also dropped, but it might be added back in a different way in the future.)

Solaris

Timekeeping in Solaris 10 is tickless. The operating system reads a hardware counter (by default, the TSC) to obtain the raw amount of time since the system booted. The wall clock time at boot is read from the CMOS time of day clock. In addition, while running, Solaris periodically checks its estimate of wall clock time against the CMOS TOD clock and uses this information to correct and refine its boot-time measurement of the TSC's running rate.

The Solaris timer callback service also does not use a periodic interrupt. Instead, it maintains a one-shot interrupt set to wake the system up in time for the next scheduled callback. This interrupt is rescheduled as timer callbacks fire, are added, or are removed.

These characteristics of Solaris are similar to Linux with the clocksource and NO_HZ features and are a good match for running in a virtual machine.

Solaris does currently exhibit two minor problems when running in a virtual machine.

First, when running on multiple processors, Solaris attempts to resynchronize the TSCs at boot time by measuring how far out of sync they are, computing a TSC offset for each processor, and applying that offset in software as a correction to each subsequent reading of the corresponding TSC. Unfortunately, while a virtual machine is booting, all virtual processors are not necessarily simultaneously scheduled on physical processors when Solaris takes its measurement, causing Solaris to measure the TSCs as being slightly out of sync and compute an unwanted offset. This sort of problem is not unique to Solaris, but other operating systems that attempt to resynchronize the TSCs do so in hardware, by writing to the TSCs, which gives the virtual machine the opportunity to detect the problem and apply a heuristic to correct it. VMware is working with Sun to disable TSC resynchronization when Solaris runs in a virtual machine.

A second small issue is that Solaris can occasionally find the CMOS TOD clock to be too far off from the value it expects to see and conclude that the CMOS clock is not working properly. This results in a harmless warning message printed to the Solaris console log.

Synchronizing Virtual Machines and Hosts with Real Time

As discussed in "Initializing and Correcting Wall-Clock Time" on page 3, for long-term accuracy, both physical and virtual machines generally need to run software that periodically resynchronizes the wall clock time maintained by the operating system to an external clock.

There are two main options available for guest operating system clock synchronization: VMware Tools periodic clock synchronization or the native synchronization software that you would use with the guest operating system if you were running it directly on physical hardware. Some examples of native synchronization software are Microsoft W32Time for Windows and NTP for Linux. Each option has some advantages and disadvantages. We discuss each briefly here and then add details in subsequent sections.

VMware Tools periodic clock synchronization has the advantage that it is aware of the virtual machine's built-in catch-up and interacts properly with it. If the guest operating system clock is behind real time by more than the known backlog that is in the process of being caught up, VMware Tools resets the clock and informs the virtual machine to stop catching up, which sets the backlog to zero. An additional advantage of VMware Tools clock synchronization is that it does not require networking to be set up in the guest. However, at this writing, VMware Tools clock synchronization has a serious limitation: it cannot correct the guest clock if it gets ahead of real time (except in the case of NetWare guest operating systems). This limitation applies only to periodic clock synchronization. VMware Tools does a one-shot correction of the virtual machine clock that may set it either backward or forward in two cases: when the VMware Tools daemon starts (normally while the guest operating system is booting), and when a user toggles the periodic clock synchronization feature from off to on.

Native synchronization software has the advantage that it is generally prepared to deal with the virtual machine clock being either ahead of or behind real time. It has the disadvantage that it is not aware of the virtual machine's built-in catch-up and thus typically does not synchronize time as well in a virtual machine as it does when run directly on physical hardware.

One specific problem occurs if native synchronization software happens to set the guest operating system clock forward to the correct time while the virtual machine has an interrupt backlog that it is in the process of catching up. Setting the guest operating system clock ahead is a purely software event that the virtual machine cannot be aware of, so it does not know that it should stop the catch-up process. As a result, the guest operating system clock continues to run fast until catch-up is complete, and it ends up ahead of the correct time. Fortunately, such events are infrequent, and the native synchronization software generally detects and corrects the error the next time it runs.

Another specific problem is that native synchronization software may employ control algorithms that are tuned for the typical rate variation of physical hardware timer devices. Virtual timer devices have a more widely variable rate, which can make it difficult for the synchronization software to lock on to the proper correction factor to make the guest operating system clock run at precisely the rate of real time. As a result, the guest operating system clock tends to oscillate around the correct time to some degree. The native software may even determine that the timer device is broken and give up on correcting the clock.

Despite these potential problems, however, testing has shown that NTP in particular behaves fairly well in a virtual machine when appropriately configured (see "Using NTP in Linux and Other Guests" on page 17). NTP is prepared for some of its readings to be anomalous because of network delays, scheduling delays on the local host, and other factors and is effective at filtering out such readings.

Generally, it is best to use only one clock synchronization service at a time in a given virtual machine to ensure that multiple services do not attempt to make conflicting changes to the clock. So if you are using native synchronization software, we suggest turning VMware Tools periodic clock synchronization off.

Using VMware Tools Clock Synchronization

VMware Tools includes an optional clock synchronization feature that can check the guest operating system clock against the host operating system clock at regular intervals and correct the guest operating system clock. VMware Tools periodic clock synchronization works in concert with the built-in catch-up feature in VMware virtual machines and avoids turning the clock ahead too far. VMware Tools also performs one-time corrections of the guest operating system clock after certain events, even if periodic synchronization is turned off.

Enabling Periodic Synchronization

To enable VMware Tools periodic clock synchronization in a guest, first install VMware Tools in the guest operating system. You can then turn on periodic synchronization from the graphical VMware Tools control panel within the guest operating system. Alternatively, you can set the `.vmx` configuration file option `tools.syncTime = true` to turn on periodic synchronization. Synchronization in a Linux guest works even if you are not running the VMware Toolbox application. All that is necessary is that the VMware `guestd` process is running in the guest operating system and `tools.syncTime` is set to `TRUE`.

By default, the daemon checks the guest operating system clock only once per minute. You can specify a different period by setting the `.vmx` configuration file option `tools.syncTime.period` to the desired time period (value specified in seconds). When the daemon checks the guest operating system clock, if it is much farther behind the host time than the virtual machine's built-in catch-up mechanism expects it to be, the daemon resets the guest operating system clock to host time and cancels any pending catch-up. As discussed above, this periodic check currently cannot correct the guest operating system time if it is ahead of the host time except in NetWare guest operating systems.

Disabling All Synchronization

It is normal for a guest operating system's clock to be behind real time whenever the virtual machine is stopped for a while and then continues running—in particular, after a suspend and resume, snapshot and revert to snapshot, disk shrink, or VMotion operation. Therefore, if VMware Tools is installed in a guest operating system, the VMware Tools daemon corrects the guest operating system clock after these events occur, even if periodic time synchronization is turned off.

Occasionally, you may need to test a guest operating system with its clock set to some value other than real time. Some examples include setting a virtual machine's date to 1999 to work around Y2K problems in legacy software or setting a virtual machine to various times to test date printing routines. You may want to have the virtual machine show the same time whenever it is powered on, to specify a constant offset from real time, or to synchronize a virtual machine with a Microsoft Windows domain controller whose time is out of sync with the host machine on which the virtual machine is running.

VMware Tools can synchronize guest operating systems only to the real time as maintained by the host operating system, so you need to disable VMware Tools clock synchronization completely if you want to maintain a fictitious time in a guest operating system.

VMware Tools automatically updates the guest operating system's time to match the host operating system's time in a few other cases in which the guest can be expected to have lost a large amount of time, even if periodic clock synchronization is turned off. To maintain a fictitious time, you need to set the following options to `FALSE`.

NOTE In some product versions, you may have to use `0` in place of `FALSE`.

- `tools.syncTime = FALSE`
- `time.synchronize.continue = FALSE`
- `time.synchronize.restore = FALSE`
- `time.synchronize.resume.disk = FALSE`
- `time.synchronize.shrink = FALSE`
- `time.synchronize.tools.startup = FALSE`

Information on these settings is also available in VMware knowledge base article 1189 (http://kb.vmware.com/kb/1189).

Table 3 shows what each option controls.

Table 3. Time Synchronization Settings in the Configuration File

Option	Effect
`tools.syncTime`	If set to `TRUE`, the clock syncs periodically.
`time.synchronize.continue`	If set to `TRUE`, the clock syncs after taking a snapshot.
`time.synchronize.restore`	If set to `TRUE`, the clock syncs after reverting to a snapshot.
`time.synchronize.resume.disk`	If set to `TRUE`, the clock syncs after resuming from suspend and after migrating to a new host using the VMware VMotion feature.
`time.synchronize.shrink`	If set to `TRUE`, the clock syncs after defragmenting a virtual disk.
`time.synchronize.tools.startup`	If set to `TRUE`, the clock syncs when the tools daemon starts up, normally while the guest operating system is booting.

Because guest operating systems generally get their time from the virtual CMOS TOD clock when they are powered on, you need to set this device to your fictitious time if you want the time to persist across guest operating system restarts.

If you want to start a guest operating system with the same time on every startup, use the `rtc.startTime` option described in "Virtual CMOS RTC" on page 7.

If, instead, you want the guest operating system to have a constant offset from real time as maintained by the host, you can use the `rtc.diffFromUTC` option, or simply set the CMOS TOD clock from the virtual machine's BIOS setup screen or from within the guest operating system. In Microsoft Windows, setting the system time automatically updates the CMOS clock. In Linux, you can use the `/sbin/hwclock` program to set the CMOS clock. Alternatively, because most Linux distributions are configured to copy the system time into the CMOS clock during system shutdown, you can simply set the system time and shut down the guest operating system before restarting it again.

Using Microsoft W32Time in Windows Guests

The Windows Time Service (W32Time), present in Windows 2000 and later, implements a simple variant of the Network Time Protocol (NTP). The subset is called SNTP. W32Time allows you to synchronize a Windows machine's clock in several different ways, each providing a different level of accuracy. Like the CMOS-based time daemon, W32Time is not aware of any attempts by a virtual machine to process timer interrupt backlogs and catch the virtual machine's clock up to real time, so corrections by W32Time can occasionally overshoot real time, especially in older versions. However, W32Time should generally correct such errors on its next resynchronization. Newer versions of W32Time (in Windows Server 2003 and later) seem to incorporate a full NTP-like algorithm and perform better, comparably to the NTP implementation used in Linux. Turning on VMware Tools periodic clock synchronization does not disable W32Time.

At this writing we do not yet have specific recommendations on how to configure W32Time for best performance in a virtual machine.

A few customers have a requirement to use a virtual machine as a W32Time server, to provide time to other systems, but do not want the virtual machine to be a W32Time client. Instead, the virtual machine gets its time using VMware Tools or runs using a fictitious time as described above. W32Time does have the ability to run in a server-only mode. For instructions on setting up W32Time to run in this mode, refer to Microsoft documentation on the Windows Time Service—specifically, the `NoSync` registry option.

Using NTP in Linux and Other Guests

The Network Time Protocol is usable in a virtual machine with proper configuration of the NTP daemon. The following points are important:

- Do not configure the virtual machine to synchronize to its own (virtual) hardware clock, not even as a fallback with a high stratum number. Some sample `ntpd.conf` files contain a section specifying the local clock as a potential time server, often marked with the comment "undisciplined local clock." Delete any such server specification from your `ntpd.conf` file.
- Include the option `tinker panic 0` at the top of your `ntp.conf` file. By default, the NTP daemon sometimes panics and exits if the underlying clock appears to be behaving erratically. This option causes the daemon to keep running instead of panicking.
- Follow standard best practices for NTP: Choose a set of servers to synchronize to that have accurate time and adequate redundancy. If you have many virtual or physical client machines to synchronize, set up some internal servers for them to use, so that all your clients are not directly accessing an external low-stratum NTP server and overloading it with requests.

The following sample `ntp.conf` file is suitable if you have few enough clients that it makes sense for them to access an external NTP server directly. If you have many clients, adapt this file by changing the server names to reference your internal NTP servers.

NOTE Any `tinker` commands used must appear first.

```
# ntpd.conf
tinker panic 0
restrict 127.0.0.1
restrict default kod nomodify notrap
server 0.vmware.pool.ntp.org
server 1.vmware.pool.ntp.org
server 2.vmware.pool.ntp.org
server 3.vmware.pool.ntp.org
```

Here is a sample `/etc/ntp/step-tickers` corresponding to the sample `ntp.conf` file above.

```
# step-tickers
0.vmware.pool.ntp.org
1.vmware.pool.ntp.org
```

Make sure that `ntpd` is configured to start at boot time. On some distributions this can be accomplished with the command `chkconfig ntpd on`, but consult your distribution's documentation for details. On most distributions, you can start `ntpd` manually with the command `/etc/init.d/ntpd start`.

Host Clock Synchronization

If you are using VMware Tools to synchronize your guest operating system clock to the host clock, or if your guest operating system initializes its time from the virtual CMOS TOD clock, it is important for your host clock to have accurate time. In addition, if you are using native clock synchronization software in the guest operating system, you might choose to use the host as a time server for the virtual machine. With either such setup, your host receives the correct time from the network, and your virtual machines receive the correct time from the host operating system.

If you are using Microsoft Windows as the host for a VMware hosted product, the Windows Time Service (W32Time) can be a good way to synchronize the host clock. There are many ways to configure W32Time, some of which give more precise synchronization than others. See Microsoft's documentation for details. In addition to W32Time, there are also many other third-party clock synchronization programs available for Windows.

If you are using Linux as the host for a VMware hosted product, NTP is a good way to synchronize the host clock. The NTP daemon is called `ntpd` on current distributions, `xntpd` on older ones.

VMware ESX and ESXi also include an NTP daemon. You can enable and configure NTP from the Virtual Infrastructure Client. For older versions of ESX, you can configure NTP using the instructions in VMware knowledge base article 1339 (http://kb.vmware.com/kb/1339).

At this writing (with the latest release of ESX being version 3.5), the ESX NTP daemon runs in the service console. Because the service console is partially virtualized, with the VMkernel in direct control of the hardware, NTP running on the service console provides less precise time than in configurations where it runs directly on a host operating system. Therefore, if you are using native synchronization software in your virtual machines, it is somewhat preferable to synchronize them over the network from an NTP server that is running directly on its host kernel, not to the NTP server in the service console. In VMware ESXi, there is no service console and the NTP daemon runs directly on the VMkernel.

Time and Performance Measurements Within a Virtual Machine

Customers often ask to what extent they can trust timing and performance measurements taken within a virtual machine and how to supplement these measurements to fully understand the performance of applications running in a virtual machine. A complete treatment of this topic is beyond the scope of this paper, but we can give some general guidance here.

Time Measurements

Time measurements taken within a virtual machine can be somewhat inaccurate because of the difficulty of making the guest operating system clock keep exact time, as discussed at length above.

There are several steps you can take to reduce this problem.

- Where possible, choose a guest operating system that has good timekeeping behavior when run in a virtual machine, such as one that uses tickless or VMI timekeeping.
- Configure the guest operating system to work around any known timekeeping issues specific to that guest version. See the VMware knowledge base for details.
- Use clock synchronization software in the guest.

There are also a number of ways to avoid using the guest operating system's clock, if you are writing or modifying software specifically to make timing measurements in a virtual machine.

- Use the feature documented in "Pseudoperformance Counters" on page 9.
- For measuring relatively long time intervals, get the starting and ending time directly from a network time server, using a program like `ntpdate` or `rdate`.
- Read the virtual CMOS TOD clock, because it runs in real time, not apparent time. However, this clock is precise only to the nearest second.

Performance Measurements

Performance measurements reported by a guest operating system running inside a virtual machine are meaningful and in some cases just as accurate as when the operating system is running on physical hardware. However, you must interpret these results with care, because the guest operating system does not have the full picture of what is happening on the host.

To get a complete picture, you also need to look at statistics taken by the VMkernel. With ESX and ESXi, VMkernel performance statistics are available from Virtual Center, from tools such as `esxtop`, and also inside the guest operating system using the VMware Guest SDK (also known as `guestlib` or `vmGuestLib`).

To get a complete picture when using VMware hosted products, use the host operating system's standard tools such as Windows `perfmon` and Linux `top`.

Considerations for several specific types of performance measurement follow.

Event Counts

Generally, event counts taken by the guest operating system are accurate, but they count only events happening in that specific virtual machine. On ESX, for example, the guest operating system's count of context switches is a perfectly accurate count of switches between guest operating system processes, but it does not count how often the VMkernel descheduled this virtual machine and ran a different one. The latter statistic is available from the VMkernel. Similarly, on hosted products, the guest operating system does not know how many times the host operating system descheduled the virtual machine and ran a different host process.

As another example, the guest operating system's count of interrupts accurately reflects how many virtual interrupts it received, but it does not count physical interrupts on the host system.

Memory Usage

Memory usage counts taken by the guest operating system accurately reflect the guest operating system's usage of its virtualized physical memory. They do not, however, show how much real physical memory the virtual machine has been able to save using VMware techniques such as ballooning, page sharing, and lazily allocating pages that the guest operating system has never touched, nor do they reflect any memory that has been swapped to disk at the host level.

CPU Usage

CPU usage measurements taken by a guest operating system generally give an approximately correct measure of the relative CPU usage of various processes running in the guest operating system, but they do not reflect how loaded the host system is, because most guest operating systems are unaware that they are running in a virtual machine and are thus time-sharing the physical hardware with other virtual machines. The following paragraphs clarify this point.

Operating systems use one of two basic mechanisms to charge and account for CPU utilization: statistical sampling or exact measurement.

The statistical sampling method is more common. With statistical sampling, whenever a timer interrupt occurs, the operating system checks what process was interrupted (which may have been the idle process) and charges that process for the full amount of time that has passed since the last timer interrupt. This charging is often incorrect, because the current process may not have been running for the amount of time reported, but over the long run, the errors average out to near zero and the method provides a useful result.

With the exact measurement method, on the other hand, the operating system uses a performance counter provided by the CPU (often the TSC) to measure the exact number of cycles that it gives to each process.

If the operating system is using the statistical sampling method and the timer device in use is running in apparent time (the most common case), the guest operating system charges all the apparent time that passes to one or another of its processes. This occurs because the virtual machine delivers all of the timer interrupts that the guest operating system has requested, but it shifts them in time so that they all occur while the virtual machine is running—none while it is descheduled. As a result, time when the virtual machine has in reality been descheduled is charged by the guest to guest processes. This charging occurs randomly, approximately in proportion to the actual CPU usage of each process. Thus the guest operating system's CPU usage statistics accurately reflect the CPU consumption of guest processes relative to one another but not the absolute fraction of a host CPU they consume. For example, if the virtual machine is getting about 50 percent of a physical CPU and has two processes each consuming about an equal amount of time, with no idle time, the guest operating system reports each process as consuming 50 percent of a virtual CPU. From a larger point of view, however, each process is actually consuming only 25 percent of a physical CPU.

The results are similar with the exact measurement method using the TSC. Time during the period the virtual machine was descheduled is rapidly caught up the next time it runs again by running the TSC faster, so the next guest operating system process to run is charged for this time. This excess charging is randomly distributed among the guest operating system processes, roughly in proportion to how much time the process is really consuming. So the guest operating system's "exact" CPU usage measurements are inflated and thus no longer exact, but on average over the long term, guest operating system CPU usage statistics do accurately reflect the CPU consumption of guest operating system processes relative to one another.

The results can be somewhat different if the guest operating system is using the statistical sampling method but with a timer device that is in "lazy" mode (see "Virtual Local APIC Timer" on page 8). In this case, if multiple virtual timer interrupts are scheduled to occur while the guest operating system is descheduled, they effectively merge into one. Thus, most descheduled time is not charged to any guest operating system process. With some guest operating systems, idle time may be computed as apparent time minus the sum of times charged to other processes, in which case descheduled time is counted as idle time. In others, the idle process is charged using statistical sampling as with other processes, so the total charged time does not add up to 100 percent of real time.

Finally, some guest operating systems can explicitly account for descheduled time. An operating system running in a virtual machine and using the VMI paravirtualization interface can obtain and display the amount of stolen time—that is, the amount of time when the kernel wanted to run a nonidle process but was descheduled. Alternatively, a Windows or Linux guest operating system that has the VMware VMdesched driver (also called the "timer sponge") installed shows most descheduled time as having been used by the `vmdesched` process instead of a real guest operating system process. VMdesched is compatible only with operating systems that use statistical sampling for process accounting and tick counting for timekeeping, and the current implementation works only on uniprocessor guests. VMdesched works by manipulating the

timing of virtual timer interrupts so that most catch-up interrupts occur while the `vmdesched` process is running. (In fact, the process does nothing more than run briefly when there are catch-up interrupts to be delivered.) Because `vmdesched` adds overhead and is not compatible with some of the newer kernel technologies, it is likely not to be further developed and may be dropped in the future.

Because of these issues, total CPU load (or, conversely, total idle time) measured from within a virtual machine is not a very meaningful number, even though CPU usage of nonidle guest operating system processes relative to one another is meaningful. Therefore, if you are running software in a virtual machine that measures and adapts to total system load, you should experiment to find out how the software behaves, and you may find that you need to modify the software's measurement and adaptation algorithms.

Resource Pressure

Because timekeeping requires some CPU resources and requires some activities to be performed in a timely way—especially when tick counting is being used—the guest operating system clock in a virtual machine that does not get enough resources can fall behind or otherwise misbehave. This section discusses some potential problems.

CPU Pressure

If the guest operating system is using tick counting and it does not get enough CPU time to handle the number of timer interrupts per second that it has requested, its clock falls behind real time.

You can deal with CPU pressure issues in either of two ways.

- Where possible, configure the guest operating system to use a lower timer interrupt rate.

  With Linux guest operating systems, choose a tickless kernel if possible, or a kernel that uses a relatively low base timer interrupt rate, or a kernel that has the `divider=` option that lets you lower the rate. See the Linux best practices guide in VMware knowledge base article 1006427 (http://kb.vmware.com/kb/1006427). Avoid configuring your virtual machines with more virtual processors than needed.

  With Windows guest operating systems, try to avoid running software in the guest operating system that raises the timer interrupt rate. See "Microsoft Windows" on page 10.

- Give more CPU time to the virtual machine.

  On both VMware ESX and VMware hosted products, avoid overcommitting the host CPU by running so many virtual machines that some or all of them cannot get enough time to handle all their timer interrupts. The exact number of virtual machines you can run depends on what applications you are running in them and how busy they are, so we cannot give a guideline in this paper. Other VMware publications are available to help with capacity planning, and features such as VMware Distributed Resource Scheduler (DRS) and VMware Distributed Power Management (VMware DPM) are useful in optimizing the placement of virtual machines on physical hosts and helping you keep the right number of physical hosts powered on to handle the current resource usage by virtual machines.

  In addition, on VMware ESX, if a specific virtual machine is not getting enough CPU time to handle all its timer interrupts, you can give it a CPU reservation to ensure it gets enough time. This comes at the expense of other virtual machines that might otherwise have been allocated that time.

Memory Pressure

Memory pressure can indirectly cause CPU pressure. You can overcommit memory on an ESX host—that is, configure the virtual machines on that host with a total of more memory than physically exists on the host—and ESX is still able to run all the virtual machines at once. ESX uses several techniques to conserve and share memory so that virtual machines can continue to run with good performance in the presence of memory overcommitment, as long as the overcommitment factor is not too high. In certain cases, memory overcommitment that is too high or not configured properly can cause timekeeping problems.

The details of ESX memory management are beyond the scope of this paper, but this following points provide a quick overview:

- When the ESX VMkernel starts a virtual machine, it does not immediately give the virtual machine enough real physical memory to back all of the virtual physical memory that it was configured to have. Instead, the VMkernel allocates memory as needed.
- The VMkernel uses page sharing. It continually scans physical memory to find cases where multiple memory pages have the same content and collapses them down into a single shared, copy-on-write page.
- The VMkernel uses ballooning. If a guest operating system has the VMware Tools `vmmemsched` (or "balloon") driver installed, whenever the VMkernel needs to take physical memory away from the virtual machine, it does so by asking the guest operating system to use its own internal paging mechanisms to swap out memory.
- As a last resort, when no other mechanisms have reclaimed enough memory, the VMkernel chooses virtual machine pages and forcibly reclaims them by copying them to a swap file at the VMkernel level.

A timekeeping issue can arise when pages have been swapped out by the VMkernel. Because the VMkernel has no insight into what the guest operating system is doing with its pages, it can sometimes swap out a page that the guest operating system will soon need. The next time the guest operating system references that page, the VMkernel has to swap the page back into RAM from disk. During the swap-in, the guest operating system stops completely. It cannot run even to handle virtual interrupts, such as timer interrupts. (In contrast, when a guest operating system swaps out its own memory using ballooning, it usually can avoid swapping out pages that will be needed again soon, and the guest operating system continues to run and process virtual timer interrupts while it is swapping memory back in.) As a result, the guest operating system clock can fall behind while pages are being read in from the VMkernel swap file. Swap-ins from disk typically take around 10ms per page, and sometimes many pages must be swapped in back to back, so the guest operating system clock can fall behind by many seconds during a burst of swapping. The clock does typically catch up fully once the guest operating system's working set has been swapped in, however.

You can avoid this memory pressure issue in either of two ways:

- Install VMware Tools in your virtual machines and ensure that memory ballooning is enabled. In most cases, ballooning is able to reclaim enough memory that swapping at the VMkernel level is not required. This is the preferred approach.
- Ensure that your virtual machines are backed by enough physical memory to avoid swapping. You can avoid the need for swapping by keeping the memory overcommit factor on your ESX hosts low or avoiding overcommit entirely. Or you can prevent swapping for a specific virtual machine by setting its memory reservation to 100 percent of its configured memory size.

Troubleshooting

This section discusses some troubleshooting techniques. For additional information, search for "time" or "clock" in the VMware knowledge base (http://kb.vmware.com).

Best Practices

The first step in dealing with timekeeping issues is preventive: check that your host and virtual machine are configured properly. To summarize the main points:

- If possible, use the most recent release of your VMware product, or at least the most recent minor release of the major version you are using. We are always working on improving timekeeping performance and fixing problems.
- If possible, use the most recent supported minor version of the guest operating system in each of your virtual machines. Updates and vendor patches sometimes fix timekeeping issues, especially in the case of Linux guest operating systems, in which the timekeeping system has been undergoing rapid evolution. Check the VMware knowledge base for articles about specific configuration options or workarounds for guest operating system bugs that may be needed for the operating system version each of your virtual machines is running. In particular, for Linux guests, see the best practices guide in knowledge base article 1006427 (http://kb.vmware.com/kb/1006427).

- Check that your host system is configured for the correct time and time zone. Check that it is running suitable clock synchronization software, as described in "Host Clock Synchronization" on page 18.
- Check that your virtual machines are set to the correct time zone. Also, for Linux guest operating systems, it is best to set the option in your Linux distribution to keep the so-called "hardware" clock (that is, the virtual CMOS TOD clock) in UTC, not local time. This avoids any confusion when your local time changes between standard and daylight saving time (in England, "summer time").
- Check that you have appropriate clock synchronization software installed and configured in your virtual machines, as described in "Synchronizing Virtual Machines and Hosts with Real Time" on page 14.
- Check that VMware Tools is installed in your virtual machines. Even if you are not using VMware Tools periodic clock synchronization, the one-time clock corrections discussed in "Using VMware Tools Clock Synchronization" on page 15 are important. In addition, the VMware Tools package includes specialized device drivers that improve overall performance of virtual machines, reducing CPU load and thus indirectly helping timekeeping performance as well.

Gathering Information

If you continue to have a timekeeping problem after checking the above best practices, the next step is to observe your system behavior carefully and gather detailed information about the problem. Many different problems can have similar symptoms or can appear similar if not observed or described clearly. Some specific things you can do are covered in the following sections.

Observe Symptoms Carefully

Note exactly how the virtual machine time differs from real time, under what circumstances the timekeeping problem appears, and how severe the problem is. Does the virtual machine show time that is ahead of real time or behind? Does the error remain constant, or does it increase or otherwise vary over time? How fast does the error change? Can you correlate occurrences of the problem with other activities that create load in the virtual machine or on the host?

Test Operating System Clock against CMOS TOD Clock

If you are running a Linux guest operating system, run the following script in the guest.

NOTE You may have to run the script as root, because `/sbin/hwclock` requires root privilege in some Linux distributions. When you run the script, capture the output to a file and include the output if you file a support request with VMware.

```
cat /etc/issue
uname –a
date
/sbin/hwclock
date
cat /proc/interrupts
sleep 10
cat /proc/interrupts
date
/sbin/hwclock
date
```

Using the output from the script, you can see which timer interrupts are in use and the frequency with which interrupts are generated. Check how much the values shown in `/proc/interrupts` change during the 10 second sleep measured by the guest. The timer interrupts most commonly used by Linux are 0 or "timer" (the PIT) and LOC (the local APIC timer).

This script also provides a rough way to observe any large difference in running rate between the virtual machine and host clocks. The `date` command returns the guest operating system clock time. The `/sbin/hwclock` command returns the CMOS TOD clock time, which VMware virtualizes at a fixed offset from the host's clock.

Turn on Additional Logging

You can turn on additional logging of timekeeping statistics in a virtual machine by adding the following lines to its `.vmx` configuration file and restarting the virtual machine:

```
timeTracker.periodicStats = TRUE
timeTracker.statInterval = 5
```

The second line specifies the sampling interval (in seconds). The default interval is 60 seconds. If you are planning to file a support request with VMware, please enable these settings, do whatever is necessary to reproduce the problem, and run the affected virtual machine in its problematic state for about 30 minutes. Include the resulting `vmware.log` file with your report.

The following listing shows the time tracker statistics output from a typical `vmware.log` file from a recent VMware product. The format of this output is subject to change.

```
Jul 30 13:16:11.044: vmx| TimeTrackerStats behind by 2246 us; running at 101%; mode 0; catchup
               limited 4855 us; 0 stops, 0 giveups, 0 numLargeBumps, maxLargeBump: 0 cycles
Jul 30 13:16:11.045: vmx| TimeTrackerStats CMOS_P 320 ints, 64.00/sec, 64.01 avg, 64.00 req; 5462
               tot, 5461 req; 1808 loprg, 70756 rtry; behind -15606 us
Jul 30 13:16:11.045: vmx| TimeTrackerStats timer0 91 ints, 18.20/sec, 18.21 avg, 18.21 req; 1645
               tot, 1644 req; 278 loprg, 14994 rtry; behind  -36140 us
Jul 30 13:16:11.045: vmx| TimeTrackerStats PIIX4PMTT 2 ints, 0.40/sec, 0.42 avg, 0.43 req; 34
               tot, 34 req; 0 loprg, 0 rtry; behind -1026844 us
```

The following points describe key phrases in this report:

- `behind by 2246 us`—the virtual machine's built-in time tracker knows that the guest operating system's clock is slightly behind real time, by 2246µs.
- `running at 101%`—the virtual machine ran the guest operating system's clock at an average of 101 percent of normal speed since the last time statistics were printed (roughly `timeTracker.statInterval`).
- `mode 0`—the mode in which the time tracker is operating. Currently defined modes include the following:
  - `0`—aggressive interrupt delivery. This is the normal mode.
  - `1`—smooth interrupt delivery. This special mode spaces interrupts out more. It is used for certain older guest operating systems that may have fragile interrupt handling code.
  - `2`—smooth interrupt delivery, with catch-up currently in progress.
  - `3`—lazy interrupt delivery. This is useful for tickless guest operating systems.
  - `4`—timer calibration mode. This is used briefly while a Linux guest operating system is calibrating timers during boot.
- `catchup limited 4485 us`—during the last statistics interval, the time tracker was sometimes not able to immediately catch up to real time because of a heuristic that limits the rate of catch-up to avoid causing problems in the guest operating system. A total of 4855µs of potential catch-up was delayed by this heuristic. This includes catch-up that was slightly delayed but did occur later in the interval, not just catch-up that was delayed beyond the end of the interval. Thus, it can be more than the amount the time tracker is currently behind, as in this example.
- `0 stops`–VMware Tools has not asked the time tracker to stop catch-up since the virtual machine was powered on. VMware Tools stops catch-up whenever it detects that the guest operating system's clock is significantly behind real time and turns the clock ahead.
- `0 giveups`–the time tracker itself has not detected that the guest operating system clock is too far behind to catch up.
- `0 numLargeBumps, maxLargeBump: 0 cycles`– there have been no large skews between apparent time on different virtual CPUs.

The remaining lines give details for specific timer devices. The `CMOS-P` line refers to the CMOS timer's periodic interrupt, the `timer0` line refers to PIT timer 0, and the `PIIX4PMTT` line refers to the ACPI PM timer. Other names that can appear on these lines include `CMOS-U` (the CMOS timer update interrupt) and `APICn` (the local APIC timer on virtual CPU *n*, where *n* ranges from `0` upward).

Taking the `CMOS-P` line as an example, the following points explain key phrases:

- `320 ints, 64.00/sec`—there were 320 virtual CMOS periodic timer interrupts delivered in the last statistics interval (five seconds in this example), or exactly 64 per second.
- `64.01 avg, 64.00 req`—there was an average of 64.01 interrupts per second over the interval since the guest operating system last reprogrammed this timer to a different rate. The guest operating system asked for 64.00 interrupts per second. In general, the average can be higher than the requested rate because of catch-up or lower because the time tracker is currently behind. In this example, the difference is insignificant.
- `5462 tot, 5461 req`—there has been a total of 5462 interrupts since the guest operating system last reprogrammed the timer, whereas there should have been 5461 at the nominal, requested rate.
- `1808 loprg, 70756 rtry`—on 1808 occasions, when the virtual machine wanted to deliver a virtual interrupt to the guest operating system, it was not safe to do so because the guest operating system had made too little progress running code since the last virtual interrupt of this type. The time tracker did a total of 70756 retries, implying that it often required multiple retries to deliver a single interrupt.
- `behind -15606 us`—how far in the past the next interrupt from this device is programmed to occur, relative to apparent time. This is normally a negative value, which means that the next interrupt is set to occur in the future. It can sometimes be positive because of various unusual conditions that force apparent time to move ahead even though a timer device has not yet delivered all of its interrupts.

If one of the timer devices in the guest operating system is not currently programmed in a periodic mode but has produced interrupts in the last interval, a different style of statistics line is logged for it, with a subset of the fields shown in the example above. The line uses the word `aperiodic` because it most commonly occurs when the device is being used in one-shot mode. An aperiodic statistics line looks like this:

```
Aug 17 10:58:21.264: vmx| TimeTrackerStats APIC0 aperiodic 12153 ints, 202.54/sec; 1092447 tot;
               322 loprg, 326 rtry
```

The following listing shows time tracker statistics output from an older product version running a different guest operating system. Many fields are similar. The differences are described below.

```
Mar 21 17:17:36: vmx| TimeTrackerStats behind by 104218351 cycles (43668 us); running at 100%; 0
               stops, 0 giveups
Mar 21 17:17:36: vmx| TimeTrackerStats APIC0 9972 ints, 997.40/sec, 1023.94 avg, 1000.49 req;
               51188 tot, 50015 req; 59 loprg, 60 rtry
Mar 21 17:17:36: vmx| TimeTrackerStats timer0 9970 ints, 997.20/sec, 1023.62 avg, 1000.15 req;
               51172 tot, 49998 req; 1395 loprg, 1400 rtry
```

- The `behind by` statistic is given in cycles (of the virtual TSC) as well as microseconds.
- There is no `mode` statistic. Instead, the `running at` statistic gives the rate at which the time tracker is currently attempting to catch up the guest clock. `100%` means that there is no catch-up in progress. A typical value when catch-up is in progress is `300%`, but the effective catch-up rate is generally much lower.

Gather VM-Support Dump

If you are submitting a support request, VMware Support also asks you to run the `vm-support` script to gather additional information about your host system and virtual machines. On Linux-hosted and VMware ESX systems, this script is named `/usr/bin/vm-support`. On Windows-hosted systems, the script is named `vmsupport.vbs` and is located in the VMware installation directory. See VMware knowledge base article 653 (http://kb.vmware.com/kb/653).

Resources

- "Collecting diagnostic information for VMware ESX Server "
  http://kb.vmware.com/kb/653
- "Disabling Time Synchronization"
  http://kb.vmware.com/kb/1189
- "Installing and Configuring NTP on VMware ESX Server"
  http://kb.vmware.com/kb/1339
- "Paravirtualization API Version 2.5"
  http://www.vmware.com/pdf/vmi_specs.pdf
- "Timekeeping best practices for Linux"
  http://kb.vmware.com/kb/1006427
- "Virtual Machine Seems Slow when Running a Particular Program (Clock Issue)"
  http://kb.vmware.com/kb/892

If you have comments about this documentation, submit your feedback to: docfeedback@vmware.com

VMware, Inc. 3401 Hillview Ave., Palo Alto, CA 94304 www.vmware.com

Revision: 20081017 Item: WP-065-PRD-02-02