vmware®

VMware

Timekeeping in VMware Virtual Machines

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 guest software. This white paper 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. The paper also describes several known timekeeping issues you may encounter and how to correct or work around them.

This document contains the following sections:

This white paper is intended for partners, resellers, and advanced system administrators who are deploying VMware products and need to understand the issues and work around potential problems that may arise in keeping accurate time on virtual machines.

Introduction

Generally speaking, PC-based operating systems keep track of time by counting timer interrupts or ticks. When the operating system starts up, it reads the current time to the nearest second from the computer's battery-backed (CMOS) real time clock or queries a network time server to obtain a more precise time. To update the time from that point on, the operating system sets up one of the computer's hardware timekeeping devices to interrupt periodically at a known rate (say, 100 or 1000 times per second). The operating system then fields these interrupts and keeps a count to determine how much time has passed.

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 Server, 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 if a virtual machine is running at the moment when one of its virtual timer interrupts is due, the virtual machine may not check for the interrupt at that moment and deliver it to the guest operating system on time. Constantly checking for pending virtual timer interrupts would introduce a substantial overhead, slowing down all virtual machines, so the VMware timekeeping implementation checks for virtual timer interrupts only occasionally — often not until the next real interrupt occurs on the host machine.

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 gets 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 guest is running too slowly, perhaps due to competition for CPU time from other virtual machines or processes running on the host machine, it may be impossible to feed the guest 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 and the time synchronization feature is enabled, the tools correct the clock reading in the guest operating system sometime within the next minute by synchronizing the guest operating 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*N context switches per second (even if all the virtual machines are idle). Virtual machines that request 1000 interrupts per second create ten times the context switching load. (Virtual machines running Microsoft Windows request 1000 interrupts per second if they are running certain applications that make use of the Microsoft Windows multimedia timer service. Linux virtual machines running

kernel 2.6, or versions of kernel 2.4 with certain vendor patches, do so as well, and they request even higher rates if running SMP-enabled kernels.)

Besides getting the correct initial time when the virtual machine is powered on and keeping track of the passage of time accurately after that, a virtual machine also needs to have its clock updated when it resumes operation after being suspended or when it reverts to a snapshot. In those cases, the virtual machine must be able to get the time updates it needs from the host, but must not be required to run in the host's time zone. For special applications, it must also be possible for a virtual machine to have its clock set to a fictitious time different from the time kept on the host.

The following sections provide more detail on what the timekeeping devices in a PC do, how standard operating systems use these devices, how VMware products virtualize the devices and support the special requirements discussed in this section, and how you can diagnose and deal with common timekeeping problems.

Review of Time and Frequency Units

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

Unit Abbreviation	Time Measurement
s	Seconds
ms	Milliseconds (1/1000 second)
us	Microseconds (10^{-6} seconds)
ns	Nanoseconds (10^{-9} seconds)
ps	Picoseconds (10^{-12} seconds)
Hz	Frequency (cycles or other events per second)
kHz	Kilohertz (1000 cycles or events per second)
MHz	Megahertz (1,000,000 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 time read 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 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 still is a useful abstraction.

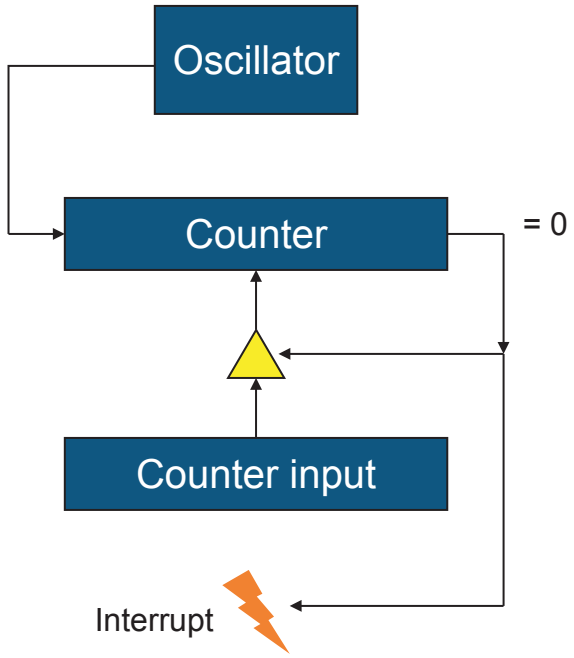

Figure 1: Block diagram of a timer device

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.)

PIT (Programmable Interval Timer)

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. Linux and most uniprocessor versions of Microsoft Windows use PIT 0 as the main system timer.

CMOS RTC (Real Time Clock)

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. Multiprocessor and ACPI-capable versions of Microsoft Windows use the CMOS periodic timer as the main system timer.

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 signal 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 (Advanced Programmable Interrupt Controller) Timers

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 Pentium and later 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 generally 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 (Advanced Configuration and Power Interface) or Chipset Timer

The ACPI timer is an additional system timer that is required as part of the ACPI specification. It has a 24-bit counter that runs 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, the counter turns back to zero after it reaches the maximum 24-bit binary value.) The ACPI timer continues running in some power-saving modes in which other timers are stopped or slowed. Some versions of Microsoft Windows read the ACPI timer to implement the `QueryPerformanceCounter` system call. Linux kernel 2.6 can use the ACPI timer to interpolate between PIT ticks.

TSC (Time Stamp Counter)

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, although this instruction is surprisingly slow on Pentium 4 chips. The 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.
- On shared-bus SMP machines, all the TSCs run off a common clock oscillator, so they can generally be synchronized with each other at startup time and thereafter treated as a single system-wide clock. This does not work on some NUMA (non-uniform memory access) multiprocessors, however, as different NUMA nodes often run off separate clock oscillators. Although the nominal frequencies of the oscillators on each NUMA node may be the same, each oscillator is controlled by a separate crystal, and the crystal precision is typically no better than 20 parts per million. In fact, on some systems (such as the IBM x440/x445 family), these clock rates are intentionally varied dynamically over a small range (2% or so) to reduce the effects of emitted RF (radio frequency) noise, a technique called spread-spectrum clocking.

Despite these drawbacks, both operating systems and application programs frequently use the TSC for timekeeping. Some versions of the Windows operating system read the TSC to implement the `QueryPerformanceCounter` system call. Many versions of Linux use the TSC to generate additional bits of precision for the `gettimeofday` system call, beyond the count of timer interrupts received. VMware products also use the hardware TSC for fine-grained timekeeping.

HPET (High Precision Event Timer)

The HPET is a device available in some newer PCs. Most PC systems do not have this device and most operating systems do not use it. The HPET has one central up-counter that runs continuously (unless stopped by software). It is usually 64 bits wide, but 32-bit implementations are permitted by its specification. The counter's period can be read from a register. The HPET provides multiple timers (three or more), each consisting of a register that is compared with the central counter. When a register 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.

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 guest software is not disrupted by anomalous time readings. In VMware terminology, the time that is visible to guests on their timer devices is called apparent time. Generally, all 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. (Note, however, that the host sound generated 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 (TOD) clock and the periodic, update, and alarm interrupts that the CMOS RTC provides. There is one exception: The alarm interrupt is not yet available in VMware ESX Server version 2.x, so attempts to set this interrupt in that environment are ignored.

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 provides a Coordinated Universal Time (UTC) value plus an offset. (UTC values are approximately equal to GMT, or Greenwich Mean Time.) The offset from UTC is stored in the virtual machine's `nvram` file along with the rest of the virtual machine's CMOS nonvolatile memory contents. The offset is needed because many guest operating systems want the CMOS TOD clock to display local time in the current 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 up, the offset is initialized, by default, to the difference of the host operating system's local time zone from UTC. If the guest writes a new time to the CMOS RTC, VMware software updates the offset.

If you want, 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 directly to the CMOS RTC.

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 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`.

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

All these choices reflect the way guest operating systems commonly use the CMOS timer 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 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 update-in-progress (UIP) bit, and the rollover of the TOD clock to the next second

do not all occur at the same moment, as they would on real hardware. In VMware products beginning with VMware Workstation 5, 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 an old version of USL Unix or Xenix that requires it. Setting this value for other guest operating systems may prevent timekeeping from working correctly.

Virtual Local APIC Timers

VMware products fully emulate the local APIC timer on each virtual CPU. The timer runs in apparent time, matching the other timer devices. Currently, the virtual APIC timer's input frequency is always 66.000MHz.

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 matches 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 first powered on, its virtual TSC is set, by default, to run at the same rate as the host TSC, but if the virtual machine is moved to a different host with a different CPU speed (using VMotion, or by suspending on one host machine and resuming on another), the virtual TSC continues to run at its original startup 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 will 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.

Running the TSC in apparent time is necessary for some guest operating systems to start up and run properly, particularly SMP-enabled versions of both Microsoft Windows and Linux operating systems. However, there are some drawbacks. Reading the TSC takes a single instruction (`rdtsc`) and is fast on real hardware, but in a virtual machine this instruction incurs substantial virtualization overhead. Thus, software that reads the TSC very frequently may run more slowly in a virtual machine. Also, some software uses the TSC to measure performance, and such measurements are less accurate using apparent time than using real time.

You can turn off virtualization of the TSC in a virtual machine. After you do this, reading the TSC in the virtual machine returns the host machine's TSC value and incurs no virtualization overhead. To turn off TSC virtualization, add the following setting to the virtual machine's configuration file:

```
monitor_control.virtual_rdtsc = false
```

Be aware that some guest operating systems may fail to start up if the virtual TSC is turned off. A possible workaround for this problem is to start up the guest operating system with the virtual

TSC turned on, suspend the virtual machine and add the configuration setting to turn off the virtual TSC, then resume the virtual machine.

It is also possible for the guest operating system to read the real TSC's value as a pseudo-performance counter using a VMware experimental feature (subject to change). This feature uses a trap to catch a machine instruction issued by the guest operating system and thus is slower than reading the TSC directly, but it does permit the virtual TSC to continue to show apparent time. To enable this feature, use the following configuration file setting:

```
monitor_control.pseudo_perfctr = true
```

You can then issue the machine instruction `rdpmc 0x10000` in the virtual machine to obtain the host TSC. This instruction is privileged unless the `PCE` flag is set in the virtual machine's `CR4` control register, so you may have to modify the guest operating system (or write a driver) to turn on this bit before you use it.

Virtual HPET

Current VMware products do not provide a virtual HPET.

Other Time-Dependent Devices

Computer generation of sounds is also time-sensitive. The sounds 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 to exhaust the supply of buffered sound information available to play. Another source of problems is that playback of MIDI sound (as well as many other forms of multimedia) 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 games do use them. These games currently are not playable in a virtual machine.

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 or VMware GSX Server.

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 HAL (hardware architecture layer), is installed. Most uniprocessor Windows installations 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 1000Hz (or 1024Hz on systems that use the CMOS timer) 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 1000Hz. This API is not used exclusively by multimedia applications. For example, the BEA WebLogic product and the Chameleon NFS client also raise the timer rate to 1000Hz.

Microsoft Windows has an additional time measurement feature accessed through the `QueryPerformanceCounter` system call. This name is a misnomer, since 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.

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.

Microsoft Windows has two built-in clock synchronization mechanisms that correct for problems such as lost ticks and off-frequency timer input oscillators.

1. 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 mostly harmless in a virtual machine, but it can conflict with VMware Tools time synchronization in some rare cases. In addition, it is possible (though rare) for the daemon to set the clock too far ahead because it is not aware that the virtual machine is in the process of catching up to real time. Windows provides a way for an application to disable this daemon, and it is planned that future versions of VMware Tools will do so when the time synchronization feature is turned on.
2. The Windows Time Service (W32Time), present in Windows 2000 and later, implements a simple subset 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 up the virtual machine's clock to real time. So, W32Time can set the clock too far ahead or otherwise be confused by the clock behavior, and it can conflict with VMware Tools time synchronization. In most cases, it's recommended that you turn the W32Time service off in virtual machines and run only VMware Tools time synchronization. (It is planned that future versions of VMware Tools will do this automatically. Also see *VMware Knowledge Base article 1318*.) If you have a requirement to use W32Time, see Guest Clock Synchronization With Non-VMware Software on page 16.

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. Because of these operations, some versions of Windows will fail to start up in a virtual machine if TSC virtualization is turned off, since the virtual machine is not allowed to change the value of the host's TSC.

Some multiprocessor versions of the Windows operating system program the local APIC timers to generate one interrupt per second, while 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, while others route this interrupt only to the primary processor and use interprocessor interrupts for scheduler time slicing on secondary processors.

Linux

Current versions of Linux use periodic timer interrupts for timekeeping and scheduling. 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 8 and some updates to Red Hat 7 used 512Hz, but later updates reverted to the standard 100Hz rate. Novell SuSE Pro 9.0 uses 1000Hz when the `desktop` command line option is provided to the kernel, and the SuSE installation program sets this option by default. Linux 2.6 kernels use a base rate of 1000Hz.

In addition to the main PIT 0 timer interrupts, SMP-capable kernels (as well as some UP kernels that are configured to enable the local APIC) also program the local APIC timer on each CPU to deliver interrupts 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 the local APIC requires a total of 2000 timer interrupts per second, while a two-CPU virtual machine requires 3000 interrupts per second. Linux attempts to stagger these interrupts so that taken together, the interrupts from all the timers are evenly spaced. (Since the rates are not exactly the same, however, this staggering does not work; in fact, the PIT and local APIC timer interrupts drift slowly in and out of phase with one another.) The Linux scheduler uses local APIC timer interrupts to time slice CPU resource usage among processes, but the interrupts play no role in keeping track of the time of day.

User applications on Linux can request additional timer interrupts from the CMOS timer using the `/dev/rtc` device. This feature is used by some MIDI software such as MusE.

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.

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.

Linux kernel 2.6 implements several different algorithms for interpolating the system time and lets you choose between them with the `clock=` kernel command line option. Unfortunately, all the available options have some drawbacks. (Information presented here is current up to kernel version 2.6.8.1.)

Two of the algorithms incorporate code that attempts to automatically detect lost ticks from non-processed timer interrupts and add extra ticks to correct for the time loss. Unfortunately, these algorithms may 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 due to a CPU that 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. So, 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 Linux kernel 2.6 algorithms for interpolating time that can be used in a virtual machine:

- 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 up to 10% have been observed when running this algorithm in a virtual machine. When run in a VMware virtual machine, SuSE Linux Enterprise Server version 9 (SLES9) uses this algorithm by default because of a SuSE-specific patch.
- 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. An unpatched kernel uses this algorithm by default when run in a VMware virtual machine.
- Option `clock=pit` is the simplest and most robust of the available algorithms. It uses only the PIT counter for interpolation. This option does not include lost tick correction code, so it does not gain time, but it does lose time when a tick is actually lost. This option is recommended for use in VMware virtual machines, together with VMware Tools time synchronization, to correct for the occasional lost ticks.

Other Operating Systems

The Be Operating System (BeOS) version 5 system clock typically runs too fast in a virtual machine. BeOS 5 bases its clock on the TSC, after measuring the TSC's speed against the PIT timer. This measurement is taken over too short an interval to achieve accuracy even on real hardware, and this deficiency is magnified in a virtual machine. Later versions of BeOS may have fixed this issue.

Increasing the Host Timer Interrupt Rate

Generally, to keep up with the timer interrupt rate requested by a guest operating system, VMware products require the host's timer interrupt rate to be at least as high as the guest operating system's interrupt rate. This is because guest code usually executes directly on the real processor. The virtual machine gets a chance to check whether the guest is due to receive a virtual timer interrupt only when some event occurs on the real processor that causes it to trap out of direct execution. Only physical timer interrupts can be relied on to occur regularly enough. It is sometimes helpful to make the host rate higher than the guest rate, as this provides more opportunities to catch up if the guest falls behind.

As previously described, the Windows multimedia timer service provides a way to increase the host timer interrupt rate, but only up to a maximum of 1000 or 1024Hz. Because of this limit, guest operating systems that use higher timer interrupt rates — in particular, many Linux 2.6 configurations — generally fail to keep up with real time when run under VMware for Windows. For workarounds to this problem, see *VMware Knowledge Base article 1420*.

Also, as previously described, Linux provides a way to generate additional timer interrupts at up to 8192Hz using the `/dev/rtc` device. (VMware Workstation and GSX Server for Linux use this mechanism when needed.) There are some cases where `/dev/rtc` interrupts are unavailable, because another application is using the device, the device is not configured, or the device does not work in your specific Linux kernel. For workarounds in these situations, see *VMware Knowledge Base article 892*.

VMware ESX Server can also increase its host timer interrupt rate as needed to service guest operating systems. VMware ESX Server 1.x versions dynamically vary the rate up to a maximum of 1000Hz per CPU. VMware ESX Server 2.x versions currently use a fixed rate of 1000Hz. As previously mentioned, this rate is not high enough for certain guest operating systems (for example, many Linux 2.6 guests), but in ESX Server 2.5 and later, you can manually increase the interrupt rate if needed. See *VMware Knowledge Base article 1518* for instructions. On ESX Server 3.x (future planned release), the default rate will be set higher and you will be able to increase the rate further if necessary.

Synchronizing Hosts and Virtual Machines with Real Time

Because of the way that timer devices in a virtual machine may fall behind real time and then catch up later, standard clock synchronization software such as the Windows Time Service (W32Time) or the Network Time Protocol (NTP) does not work well when run in a virtual machine. If the virtual machine is aware that it is behind real time and is already delivering timer interrupts at a higher rate so that the guest clock can catch up to real time, running non-VMware clock synchronization software inside the guest at the same time may also advance the virtual machine's clock, causing it to end up ahead of real time. Also, the widely varying timer interrupt rate of a virtual machine, as compared with real time, is likely to confuse algorithms in non-VMware clock synchronization software that attempt to detect the machine's exact timer oscillator input frequency and correct for small variations from the specified frequency.

Instead of running non-VMware clock synchronization software in virtual machines, it is recommended that you run such software in the host operating system (for VMware hosted products) or the service console (for VMware ESX Server), then run VMware Tools in each guest operating system with the time synchronization option turned on. With this setup, your host receives the correct time from the network, and your virtual machines receive the correct time

from the host operating system. Some customers may have requirements to run non-VMware clock synchronization software in guest environments, however. See Guest Clock Synchronization With Non-VMware Software on page 16, if you have this requirement.

Host Time Synchronization

On Microsoft Windows 2000 and later, the Windows Time Service (W32Time) comes with the operating system. 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.

All Linux distributions come with an NTP (network time protocol) daemon. The daemon is called `ntpd` on current distributions, `xntpd` on older ones. NTP is an excellent way to synchronize a Linux host machine's time.

VMware ESX Server also includes an NTP daemon for use on the service console. See *VMware Knowledge Base article 1339* for instructions on setting it up.

Guest Time Synchronization With VMware Tools

VMware Tools includes a time synchronization feature that periodically checks the guest operating system clock against the host operating system clock and corrects the guest clock. Unlike non-VMware synchronization software, VMware Tools time synchronization works in concert with the built-in catchup feature in VMware virtual machines and avoids turning the clock ahead too far.

To enable VMware Tools time synchronization in a guest, first install VMware Tools in the guest operating system. Next, check that time synchronization is turned on. You can enable synchronization from the graphical VMware Toolbox application within the guest. Alternatively, you can set the `.vmx` configuration file option `tools.syncTime = true` to enable time synchronization. Note that time 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 and `tools.syncTime` is set to `true`.

VMware Tools time synchronization is designed to be a second line of defense to deal with special cases where a guest operating system's clock falls behind real time despite the built-in catchup mechanism provided in the virtual machine. It is normal for a guest'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/resume, snapshot, disk shrink, or VMotion operation. These are the main cases that VMware Tools time synchronization is meant to handle. The guest's clock may also fall behind in less common circumstances, such as under heavy load when the guest has not been able to get enough CPU time to handle all its timer interrupts.

The VMware Tools time synchronization daemon is quite simple and has a few limitations. The daemon checks the guest clock only once per minute. If the guest clock is much farther behind the host time than the virtual machine's built-in catchup mechanism expects it to be, the daemon resets the guest clock to host time and cancels any pending catchup. For most guest types, the daemon never turns the guest clock backward, even if the guest's clock time is running ahead of real time. Turning the clock backward is seldom needed and can cause some guest software to become confused. If your guest's clock is running ahead of real time, see Known Issues and Troubleshooting on page 18 for troubleshooting tips and potential solutions and workarounds.

Note: Future VMware products (tentatively, Workstation 5.5, ESX Server 3.0, and GSX Server 4.0) will allow changing how often the tools daemon checks the guest time against the host time. The default period between time checks is 60 seconds. To select a different period, set the configuration variable `tools.syncTime.period` to the desired time period (value specified in seconds).

Keeping a Fictitious Time In a Guest System

Occasionally you may have a need to test a guest 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 turn off VMware Tools time synchronization if you want to maintain a fictitious time in a guest system.

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

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

Note: Information on these settings is also available in *VMware Knowledge Base article 1189*.

Here is what each option controls:

- `tools.syncTime` — If set to TRUE, the time syncs periodically, as described above.
- `time.synchronize.continue` — If set to TRUE, the time syncs after taking a snapshot.
- `time.synchronize.restore` — If set to TRUE, the time syncs after reverting to a snapshot.
- `time.synchronize.resume.disk` — If set to TRUE, the time syncs after resuming from suspend.
- `time.synchronize.shrink` — If set to TRUE, the time syncs after defragmenting a virtual disk.

Since 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 restarts. If you want to start a guest with the same time on every startup, use the `rtc.startTime` option described in the earlier section Virtual CMOS RTC. If instead you want the guest 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 time 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 directly. Alternatively, as most Linux distributions are configured to copy the system time into the CMOS TOD clock during system shutdown, you can simply set the system time and shut down the guest system before restarting it again.

Guest Clock Synchronization With Non-VMware Software

If you must run non-VMware clock synchronization software (such as the Windows time service W32Time) in a virtual machine, the built-in clock catchup that the virtual machine performs can confuse the non-VMware clock synchronization software. This confusion may cause time in the guest to get ahead of real time and generally may cause the clock synchronization software to have difficulty in tracking real time closely. Also, if VMware Tools time synchronization is enabled, both VMware Tools and the non-VMware clock synchronization software you are running will try to correct the clock without knowledge of each other, causing similar problems.

Some customers have a requirement to use a virtual machine as a primary domain controller for a Windows network. The primary domain controller must run W32Time as a time server, to provide time to secondary domain controllers and other hosts on the network. However, the domain controller does not need to use W32Time's client functionality to receive time from another source and synchronize the virtual machine's own clock. So, you can use VMware Tools to synchronize the virtual machine's clock while still running W32Time in a server-only mode. For instructions on setting up W32Time this way, refer to Microsoft documentation on the *Windows Time Service*; specifically, the NoSync registry option.

If this approach is not applicable to your situation and you must synchronize a virtual machine's clock using W32Time or other non-VMware software, you can take the following actions to minimize problems:

1. Completely disable VMware Tools time synchronization, as described in Keeping a Fictitious Time In a Guest System on page 15.
2. In addition, if you still observe the virtual machine getting ahead of real time, you can try limiting the built-in clock catchup. Normally, the built-in catchup is active whenever the guest is between 50 milliseconds and 60 seconds behind real time, and the guest clock attempts to run 200% faster than normal speed while catching up. You can modify this behavior by setting the following options in the virtual machine's `.vmx` configuration file.

   ```
   timeTracker.catchupPercentage = 200
   timeTracker.catchupIfBehindByUsec = 50
   timeTracker.giveupIfBehindByUsec = 60000000
   ```

Note: The option settings shown in the example are the default values. Reducing the `giveupIfBehindByUsec` option value may help in limiting the built-in catchup operation. Setting this option to a lower value makes the catchup give up more quickly if the virtual machine gets significantly behind real time, thereby letting the non-VMware clock synchronization software you are running take care of synchronizing the clock. It is probably not a good idea to completely disable the built-in catchup, however, since the virtual machine may then lose time faster than your operating system or application software expects to be possible in a real machine.

Time Measurements Within a Virtual Machine

Customers often ask how to measure the system CPU load from within a virtual machine, or are puzzled about seeing anomalous readings from CPU usage measurement tools running within a virtual machine. This is a difficult issue for two reasons.

First, CPU load and usage measurement tools running within a virtual machine can observe activity only within the virtual machine. But the virtual machine itself is a set of processes that are scheduled by the host operating system and receive only a varying fraction of the host CPU. Moreover, CPU load and usage measurement algorithms generally depend on precise time measurements, enough so that the distortion of time that takes place in a virtual machine can have a sizable effect.

Second and more fundamentally, it is not even clear what it should mean to measure CPU load and usage within a single virtual machine. As a simple example, suppose a virtual machine is running just one process that repeatedly computes for one second, then sleeps for one second. While the process is sleeping, the guest operating system has nothing to do (other than field timer interrupts) and spends almost all of its time in a halted state. If this were a real machine, the process's CPU usage, as well as the total CPU load, obviously would be 50%. However, a virtual machine deschedules itself whenever the guest operating system halts and does not receive any physical CPU time until the halt state ends. Thus, the process receives 100% of the physical CPU time that is actually allocated to the virtual machine, though still only 50% of the time that could potentially have been allocated to it. The situation becomes more complicated if the host machine is heavily loaded. In that case the process may receive far less than 50% of a physical CPU.

Because of these issues, CPU load or idle time measured from within a virtual machine currently is not a very meaningful number. If you are running software in a virtual machine that measures and adapts to 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.

Experiments have shown that the relative CPU usage of different processes running within a virtual machine (as measured by the guest operating system) is usually approximately correct. However, these measurements can be biased if a particular process frequently causes the virtual machine to be descheduled and thus require timer interrupt catchup when the virtual machine runs again. This bias arises because guest operating systems often use statistical sampling to determine CPU usage, with samples collected from the timer interrupt handler.

Similarly, because time in a virtual machine may frequently fall behind and catch up, measuring short time intervals within a virtual machine tends to produce inaccurate results. See Virtual TSC on page 8 for information on how to access the real TSC from within a virtual machine.

If possible, use host CPU (and other resource) usage statistics instead of statistics measured inside a guest. For VMware ESX Server, see Knowledge Base article *1078*. Statistics are also available from VMware Virtual Center.

Known Issues and Troubleshooting

This section describes some known timekeeping issues and problems and discusses some workarounds and troubleshooting techniques. For additional information, search for "time" or "clock" in the *VMware Knowledge Base*.

Gathering Information

The first step in dealing with a timekeeping problem 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 the following:

1. Check whether your host machine (or for VMware ESX Server, the Service Console) has the correct time and is running proper clock synchronization software, as described in Host Time Synchronization on page 14. If not, correct this before going on.
2. Check whether VMware Tools is installed in the guest systems that are having timekeeping problems, and whether VMware Tools time synchronization is turned on. (Of course, in some cases you may have reason not to run VMware Tools time synchronization software; if so, check and verify that you are not running it.)
3. Note exactly how the guest time differs from real time, under what circumstances the timekeeping problem appears, and how severe the problem is.
4. If you are running a Linux guest, run the following script in the guest:

Note: You may have to run the script as root, as `/sbin/hwclock` requires root privilege with some Linux distributions. When you run the script, capture the output to a file and include the output if you file a support request (SR) 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 guest and host clocks. While the `date` command returns the guest operating system clock time, `/sbin/hwclock` returns the CMOS TOD clock time, which VMware virtualizes at a fixed offset from the host's clock.

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

   ```
   timeTracker.periodicStats = TRUE
   timeTracker.statInterval = 10
   ```

   The second line specifies the sampling period (in seconds); the default period 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 output from a typical `vmware.log` file. Note that the format of this output is subject to change:

   ```
   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 phrase "`behind by 104218351 cycles (43668 us)`" means that the virtual machine's built-in time tracker knows that the guest's clock is slightly behind real time, by 43668 microseconds (=0.043668 seconds), or 104218351 CPU cycles on the guest TSC.

   The phrase "`running at 100%`" indicates that the virtual machine is currently running the guest's clock at normal speed. 300% is another common speed you may see in the vmware.log file, used when the guest clock is far enough behind that it needs to be run faster to catch up.

   The phrase "`0 stops`" means that VMware Tools has not asked the time tracker to stop catchup since the virtual machine was powered on. VMware Tools stops catchup whenever it detects that the guest's clock is significantly behind real time and turns the clock ahead.

   The phrase "`0 giveups`" means that the time tracker itself has not detected that the guest clock is too far behind to catch up.

   The "`APIC0`" line gives details for the local APIC timer on CPU 0, and the `timer0` line for PIT timer 0. Other names that can appear on these lines include `CMOS-P` and `CMOS-U` (the CMOS timer periodic and update interrupts), and `PIIX4PMTT` (the ACPI timer).

   The phrase "`9972 ints, 997.40/sec`" indicates that there were 9972 virtual APIC0 timer interrupts delivered since the last point when timeTracker statistics were output (roughly the specified `statInterval`, 10 seconds in this example), or 997.40 per second.

   The phrase "`1023.94 avg, 1000.49 req`" means that there was an average of 1023.94 interrupts per second over the interval since the guest last reprogrammed this timer to a different rate. The guest asked for 1000.49 interrupts per second. In this case, the average is higher than requested, probably because the virtual machine was in catchup mode at the point when the guest last reprogrammed the timer, so it ran fast for a while.

The phrase "`51188 tot, 50015 req`" indicates there have been a total of 51188 interrupts since the guest last reprogrammed the timer, while there should have been 50015 at the nominal, requested rate. Again the excess interrupts are probably due to catchup mode.

The phrase "`59 loprg, 60 rtry`" indicates that on 59 occasions, when the virtual machine wanted to deliver a virtual interrupt to the guest, it was not safe to do so because the guest had made too little progress running code since the last virtual interrupt of this type. The time tracker did a total of 60 retries, so usually it was able to deliver the interrupt on the first retry.

6. With VMware ESX Server, you can find out what timer interrupt rate a guest is requesting even when TimeTrackerStats are not enabled. Read the contents of the node `/proc/vmware/timers` in the Service Console file system and look for lines containing the word "guest". Here is an example:

```
       deadlineTS  periodTS  periodUS  function    data     flags
 1686744858945268    747574       500    47415c       0  periodic
 1686744870510526  14951486     10000    47b250  9dccb8  one-shot
 1686744860003050  14951486     10000    47b250  9e8310  one-shot
 1686744867214924  14972418     10014    443ba4      b7  periodic, guest 183
 1686744869355698  14951486     10000    47b250  a02c88  one-shot
 1686744873607566  14951486     10000    47b250  a0a618  one-shot
 1686744859794360   1499634      1003    443ba4      ad  periodic, guest 173
 1686744868798312  14972418     10014    443ba4      ab  periodic, guest 171
```

The line that ends "`guest 183`" indicates that the guest that has a virtual CPU with VMkernel worldID 183 is requesting a timer interrupt every 10014 microseconds; that is, at a frequency of 1000000/10014 = 99.86 Hz. This is a typical rate for a uniprocessor Windows guest. The `periodUS` shown in this line reflects the aggregate interrupt rate over all timers in the virtual machine; so, for example, an SMP Linux 2.6 guest with two CPUs would show a `periodUS` of approximately 333 — because it requests 1000 PIT interrupts per second, plus 1000 local APIC timer interrupts per second per CPU. There are several ways to find which virtual machine corresponds to each specific worldID. First, the node `/proc/vmware/sched/cpu` contains a line for each active worldID, and for virtual machine worlds, the "`name`" column contains a short form of the virtual machine's name. Alternatively, you can look for a line with the corresponding number in the `vmware.log` file for each of your virtual machines; for example:

```
Apr 21 15:04:24: vcpu-0| VMMon_Start: vcpu-0: fd=15 worldID=183
```

7. If you are submitting a support request (SR), 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 Server systems, this script is named `/usr/bin/vm-support`. On Windows-hosted systems, the script is named `vm-support.vbs` and is located in the VMware installation directory. See the *VMware Support Web page* for more information.

Specific Timekeeping Issues and Problems

This section lists specific timekeeping issues and problems along with suggested workarounds or solutions.

Guest time is wrong but time runs correctly after you reset it

If you see this issue, the possible causes and actions to take are the following.

1. Was the guest started in the wrong time zone? Check that both host and guest are set to the time zone you want, correct the time manually, and check if the problem recurs. For Linux guests, it is best to set the option in your Linux distribution to keep the so-called "hardware" clock (actually 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").
2. Is VMware Tools installed and time synchronization enabled? VMware Tools can correct the guest clock after a suspend/resume, disk shrink, or migration operation.
3. Could the guest have been started with its clock ahead of real time, or could it have been set ahead of real time manually or by some software running in the guest? VMware Tools for most guest operating systems currently does not turn the guest clock back because time going backward can confuse some software running in guests. Correct the time manually and check if the problem recurs.

Guest time runs slower than real time

If you set the guest clock correctly, but it steadily falls behind again afterward, check the following.

1. Is VMware Tools installed and time synchronization enabled? Guest time ideally should run at the correct rate even if VMware Tools time synchronization is not used, so even if turning it on helps, you may want to continue down this checklist to further investigate the problem.
2. Check for excessive load on the host. If your virtual machine is not getting enough CPU time to handle all the timer interrupts it has requested, it will fall behind real time. In this case you may see a high value for "`loprg`" in the TimeTracker statistics described in the previous section Gathering Information.
3. In general, the faster the timer interrupt rate the guest requests, the more overhead the interrupts cause and the more difficult it is for the system to deliver the interrupts rapidly enough for the guest to keep up with real time. If the guest requests a faster rate than the host operating system normally provides, VMware products attempt to increase the host timer rate dynamically, but sometimes this is not possible. It is sometimes helpful to make the host rate higher than the guest rate, as this provides more opportunities to catch up if the guest falls behind.
   - On Linux hosts, VMware products use the `/dev/rtc` device to request additional timer interrupts when needed, but on some hosts `/dev/rtc` is not available or is not configured to be able to generate periodic interrupts. In this case, if `/dev/rtc` is needed, you should see a hint popup (if hints are enabled), and there should be a message in `vmware.log` stating that `/dev/rtc` was not available. See *VMware Knowledge Base article 892* for more information and workarounds.
   - On VMware ESX Server 2.x, the VMkernel hardware timer interrupt rate is set to 1000Hz, by default. You can raise this rate to help deal with guests that need virtual timer interrupts at 1000Hz or more. Setting the rate higher will increase VMkernel overhead

slightly, but is otherwise generally harmless. See *VMware Knowledge Base article 1518*. On ESX Server 3.x (future planned release), the default rate will be set higher and you will be able to increase the rate further if necessary.

- Linux kernel 2.6 guests normally request 1000 PIT 0 timer interrupts per second, plus, in some cases, 1000 local APIC timer interrupts on each virtual CPU. For single processor guests, you can usually eliminate the unneeded APIC timer interrupts by including the kernel command line flags `noapic nolapic nosmp`. (All three flags may not be needed, depending on your exact kernel version, but it should be harmless to give all three.)

  In addition, you can recompile your guest Linux kernel to reduce the base timer rate back to 100Hz; see *VMware Knowledge Base article 1420* for more information.

4. A known bug that is fixed in recent VMware releases can cause frequent lost ticks, especially at guest timer interrupt rates of 1000Hz or higher. Upgrading to the latest release will help if you have this problem. The problem is fixed in VMware Workstation 5 and VMware ESX Server 2.5.1, and a fix is scheduled to appear in VMware GSX Server 3.2.
5. Newer Linux 2.6 kernels (beginning at least as far back as 2.6.8.1) trigger another known bug that is fixed in recent VMware releases. This bug causes the guest operating system clock to run too slowly and disables catchup. Typically the clock runs at around 50% of normal speed. A fix is currently scheduled to appear in VMware Workstation 5.5, VMware ESX Server 2.5.2, and VMware GSX Server 3.2.
6. Your VMware Workstation or VMware GSX Server host system may have power-saving technology that varies the CPU clock speed. Such technology is most common on laptops, but it is coming into use on other types of host machines as well. A varying CPU clock speed can cause virtual machines to lose or gain time. See VMware Knowledge Base articles *1227* (for Windows hosts), *1591* and *916* (for Linux hosts), and *708* (for other related tips and information).
7. In rare cases, it may be possible for VMware hosted products to measure the host CPU speed incorrectly at driver load time, even if the host machine does not have power-saving features. You can check the "`KHZEstimate`" line logged in the virtual machine's `vmware.log` file to see the estimated speed of your host, measured in KHz. In these cases, restarting the host system should eliminate the problem.

Guest time runs faster than real time

If you set your guest clock to the correct time but it steadily gains time thereafter, check the following.

Note: This problem is much less common than losing time, but there are still a few possible causes.

1. As described in the Linux section of Timekeeping in Specific Operating Systems on page 9, Linux 2.6 kernels have code that tries to automatically detect lost ticks (from unprocessed timer interrupts) and add extra ticks to correct for the time loss, but this code can overcorrect and cause a virtual machine's clock to run too fast. In some cases, the guest clock may run as much as 10% faster than real time. See *VMware Knowledge Base article 1420* for more information and workarounds for this problem.
2. Try not to run more than one form of clock synchronization software in a guest system at the same time. As discussed in Synchronizing Hosts and Virtual Machines with Real Time on page 13, if you do this, when the clock falls behind, multiple clock synchronization programs may all turn it ahead, causing it to overshoot real time. In particular, this can

occur if the Windows time service (W32Time), the built-in Windows time daemon, or some other third-party clock synchronization program is running and turns the clock ahead while the virtual machine's built-in catchup is also running. The built-in catchup cannot tell this has occurred, so it continues to run the clock at high speed and eventually advances the clock time beyond real time. Similarly, if you have both VMware Tools time synchronization and a third-party clock synchronization program running, they may both add time to the clock, causing it to get ahead of real time. However, also as discussed earlier, the built-in catchup in a virtual machine and VMware Tools are designed to work together to keep the clock correct.

If you do need to run third-party clock synchronization software in a guest, refer to Guest Clock Synchronization With Non-VMware Software on page 16.

3. If you run VMware GSX Server (or VMware Workstation) on a NUMA machine, such as an IBM x440, x445, or x460 with more than one CEC (NUMA node), several problems can arise because the hardware timestamp counters (TSCs) of the different NUMA nodes are not synchronized with each other. In fact, they run at substantially different speeds, varying by as much as 1.5%. See *VMware Knowledge Base article 1236* for more information and workarounds for this situation. Note that this workaround is currently available only on Microsoft Windows hosts in VMware GSX Server 3.1, but is scheduled for Linux 2.6 hosts in GSX Server 3.2. An automated form of the workaround is also in development.
4. Your VMware Workstation or VMware GSX Server host system may have power-saving technology that varies the CPU clock speed, which can cause virtual machines to either lose or gain time. This issue is most common on laptops, but can also arise on other types of hosts. See VMware Knowledge Base articles *1227* (for Windows hosts), *1591* and *916* (for Linux hosts), and *708* (for other related tips and information).
5. In rare cases, it may be possible for VMware hosted products to measure the host CPU speed incorrectly at driver load time even if the host does not have power-saving features. You can check the "`KHZEstimate`" line logged in the virtual machine's `vmware.log` file to see the estimated speed of your host, measured in kHz. In these cases, restarting the host machine should eliminate the problem.

Second hand on clock applet in guest moves too fast or too slowly

If you watch the clock applet running in a virtual machine, you may see its second hand move more slowly than real time for a while, then move more quickly to catch up. In general, this behavior is normal and should not be considered a problem if the time is maintained accurately over the long term. However, if the behavior is disruptive to your application, refer to the next troubleshooting item for remedies.

MIDI (or other media) plays at varying rate

When listening to MIDI music, or watching or listening to other multimedia content in a virtual machine, you may notice songs playing too slowly for a while, then playing faster to catch up. This phenomenon is typically caused by the virtual machine's clock falling behind and catching up, since some media players (particularly MIDI players) use the guest operating system's clock to determine when to play notes. This issue is difficult to resolve completely; real time media is simply a difficult application to handle well in a virtual machine. There are a few things, however, that you can try to improve the situation:

1. Make sure your host system is not overloaded. Run the virtual machine on a faster host system if necessary.

2. See the workarounds listed for the troubleshooting item Guest time runs slower than real time on page 21.
3. Search for "sound" in the *VMware Knowledge Base* and read the corresponding articles.
4. You can experiment with the timeTracker parameters described in Guest Clock Synchronization With Non-VMware Software on page 16. In some cases increasing `catchupPercentage` (to make catchup faster) or reducing `catchupIfBehindByUsec` (to trigger catchup sooner) may help, but both options are a last resort.

On a Linux host, the vmware-rtc kernel thread uses too much CPU time

When one or more of the virtual machines on a Linux host machine requests timer interrupts at a higher rate than the host's hardware timer interrupt rate, VMware products attempt to use the `/dev/rtc` device to obtain additional timer interrupts on the host machine. The vmware-rtc kernel thread fields these interrupts and uses them to wake up virtual machines as needed. In some cases, this thread can consume considerable CPU time. There are several ways of dealing with this issue.

1. If the CPU consumption is not actually causing a performance problem on your host, simply ignore it.
2. In some cases the guest timer interrupt rate can be reduced. For Linux kernel 2.6 guests, see the troubleshooting item Guest time runs slower than real time on page 21.
3. See Increasing the Host Timer Interrupt Rate on page 13 for a way to increase the base host timer interrupt rate, which may eliminate the need for extra clock interrupts from `/dev/rtc`.
4. You can prevent `/dev/rtc` from being used. This will generally cause clocks to run slow in any virtual machines you have that need the additional interrupts, but that may be acceptable to you, depending on your application. To do so, add the following setting to each virtual machine's `.vmx` configuration file, or add the setting globally to the host's configuration file (`/etc/vmware/config`):

   ```
   host.useFastClock = FALSE
   ```

   This setting should work on all recent versions of VMware for Linux, but if you have an older version on which it does not work, an alternative workaround is to remove the `/dev/rtc` node from your host filesystem. (That is, become root and run the command:

   ```
   rm /dev/rtc
   ```

VMware is tracking this issue internally and we hope to reduce the CPU usage of the vmware-rtc thread in the future.

Linux error: "IO-APIC + timer doesn't work!"

Occasionally a Linux guest will fail to start up, displaying a garbled screen image or the following error message:

```
"IO-APIC + timer doesn't work! pester mingo@redhat.com"
```

or

```
"IO-APIC + timer doesn't work! Try using the 'noapic' kernel
parameter"
```

The virtual machine's vmware.log file also includes the message:

```
"Possible PR 17486. Discarding interrupt".
```

This problem occurs because of some extremely time-critical code in the Linux startup sequence that is run if the Linux kernel is configured to use the IO-APIC (advanced programmable interrupt controller) hardware for interrupt routing. This problem occurs only during startup, not during normal operation of the virtual machine.

A simple workaround is to restart the virtual machine; the machine normally succeeds in starting up on the next try.

Another workaround is to give the guest kernel the command line parameters `noapic nolapic nosmp`. All three flags may not be needed, depending on your exact kernel version, but it is safe to provide all three. Adding kernel command line parameters is done by editing the guest's `lilo.conf` or `grub.conf` file as appropriate; see your Linux distribution's documentation if you are not familiar with this process. This workaround cannot be used if your virtual machine is configured to have more than one virtual processor; that is, if it is an SMP virtual machine. Uniprocessor virtual machines running on an SMP host or ESX Server system can use the workaround.

Limited date range

Various issues can arise in host operating systems, guest operating systems, the guest BIOS, VMware Tools, or the VMware application itself, if the host or guest date is set outside the range 1981 to 2037. VMware is in the process of correcting the upper limit values that are under our control; obviously we have no control over limitations in host and guest operating systems.

Crashes running hosted products on NUMA systems

In addition to the guest clock speed problems previously described, hosted virtual machines may crash with various assertion failures if run on a NUMA system (such as IBM x440, x445, or x460) without applying the workarounds discussed in *VMware Knowledge Base article 1236*. This issue does not apply to VMware ESX Server.

Conclusion

Timekeeping in virtual machines is a complex subject. We hope this white paper has provided information to help you understand more clearly the issues involved, the problems that can arise, and the solutions that are currently available in VMware products.

VMware, Inc. 3145 Porter Drive Palo Alto, CA 94304 www.vmware.com
 Revision: 20050725 Item: NP-ENG-Q305-127