CHAPTER 6

Operating Systems

Software computation and data manipulation is the essence of the work done on a server and in a datacenter. What is often overlooked is the critical role operating systems play in determining both the performance (the speed of work) and the cost (the energy consumed) doing that work. Operating systems manage a software work plan; an efficient work plan gets work done faster and at lower cost.

Well-optimized operating systems can increase the time spent in low-power CPU, memory, and interconnect states. They avoid careless use of system resources that can limit a server's use of low-power states—for example, polling for completion of some event or activity, choosing a suboptimal resolution of the system timer, or performing operations that only utilize a small subset of logical processors in the system. Well-optimized operating systems adapt their use of power management features to match changes in workloads, changes in server utilization, and changes in operator preferences.

This chapter begins with an overview of operating systems and their role in selecting power states, scheduling, and memory management. This includes a description of the interfaces operating systems use for power state control as well as the policies and metrics the OS uses to drive those decisions. Additional considerations for virtualized environments are discussed, including consolidation and virtual machine migration. The chapter concludes with a comparison of different operating environments, the unique power management capabilities of these environments, and how these capabilities have changed over time.

Note The term *operating system* (OS) used throughout this chapter is inclusive of both a native operating system and a virtual machine monitor (VMM). A VMM assumes the same power management roles and responsibilities that a native OS does, so information in this chapter applies equally to both environments.

Operating Systems

Once equipped with power state information and the appropriate controls from BIOS, the OS must implement power management policies that determine how and when to use various power states based on system activity or any known power and performance constraints. Striking the right balance between low power and high performance is a difficult problem since the right balance differs greatly from environment to environment and user to user. Operating system power management (OSPM) policies may be unaware of service-level agreements in place or of quality of service requirements. They may not understand whether the system is running a stand-alone application or whether it is part of a complex distributed application. Even knowledge of specific applications executing tells the OS very little about what the appropriate balance is between power and performance. To an OS, a web server hosting the front end for a family photo sharing service looks identical to a web server hosting the front end for critical financial transactions.

The diversity of power and performance requirements in a datacenter presents a unique challenge for implementing and optimizing OSPM policy on servers. In other compute environments, such as phones and tablets, common performance expectations are known ahead of time. For example, gaming is good at 30 frames per second and great at 60 frames per second. Holding a device in your hand shouldn't make your hand perspire, touch gestures should be processed in a matter of milliseconds, and creating an audible pause during audio playback is unacceptable.

In a server environment, the performance impact of power management features varies greatly based on the workload, application, system configuration, and performance requirements. Some workloads will see no performance impact whereas others may see substantial impact. In the absence of known performance requirements, OSPM policy needs to make the best decisions it can regarding the balance between low power and high performance using the limited information it does have. The two primary policies the OS is responsible for is the selection of C-states and P-states. The following sections describe the mechanisms these policies use for hardware state control as well as monitoring capabilities and metrics used for policy feedback and decision making.

C-state Control

As indicated in Figure 6-1, most operating systems implement OSPM policies in device drivers. The processor can be thought of as just another device in this context. The drivers communicate with hardware using a set of standard control interfaces that are described by ACPI or ascertained with CPUID. When an OS has no work to be done on a logical processor, it enters an idle function where eventually one of two instructions can be executed to transition the logical processor into an idle C-state.

ACPI c-state	Hardware C-state	Number of Sub-states	Description	ECX [bit 0] Treat interrupt as break event even if masked?	EAX [bits 7:4] Target State?	EAX [bits 3:0] Targrt Sub-state?
C1	C0	1	Skip MWAIT	0/1	1111	0
C1	C1	2	C1 - Halt execution	0/1	0	0
C1			C1E - Halt execution, then drop core voltage and frequency	0/1	0	1
C2	C3	1	C3 - Halt execution, flush core caches	0/1	1	0
C2	C6	1	C6 - Halt execution, flush core caches, power gate core	0/1	10	0

Figure 6-1. *Example MWAIT hints OS uses to specify various processor C-states*

MWAIT

The most common way for an OS to enter an idle state is by executing the MWAIT instruction. MWAIT can only be executed in Ring 0, so applications are unable to directly enter idle states. MWAIT provides the OS control over the specific C-state and sub-state to enter and under what conditions that C-state can be exited. The OS conveys C-state control details through two general-purpose registers used by MWAIT.

The OS uses the ECX register to specify how it wants C-states to exit upon external interrupts. When the OS-specified interrupts should be treated as break events, execution resumes within the idle function rather than directly executing an interrupt service routine (ISR). The OS uses the EAX register to specify the target C-state (C1, C3, or C6) and the target sub-state. The MWAIT sub-state is used for lower-level control of a specific C-state. For example, if a specific C-state flushes the caches upon C-state entry, different sub-states may be used to modify the behavior of the cache flush, such as flushing the entire cache at once, or flushing the cache progressively over multiple time-delayed phases. Figure 6-1 is an example of various C-states supported on a processor and the MWAIT hints used to specify these states.

Any interrupt—such as a timer interrupt, device interrupt, or inter-processor interrupt (IPI)—will cause a C-state to exit and execution to resume. In addition to C-states exiting based on interrupts, MWAIT can be used in combination with a MONITOR instruction. Executed by the OS before the MWAIT instruction, a MONITOR instruction allows the OS to specify a memory address for the processor to monitor. With a MONITOR instruction armed, any subsequently entered C-state will be exited if the monitored data address is modified. This provides an additional mechanism for software defined C-state exit conditions. For example, the Linux kernel utilizes this mechanism by arming MONITOR/WAIT pairs with a kernel `need_resched` flag. This flag is modified by the OS to indicate that the kernel scheduler should be activated on a particular logical processor. Exiting C-states based on a modified MONITOR address also provides a mechanism for waking up a specific logical processor without needing to generate an IPI.

HLT

Older operating systems, or operating systems with deep C-states disabled, may execute the HLT (halt) instruction in place of an MWAIT in an idle function. HLT does not require, nor does it support, the same level of control that MWAIT does. HLT simply places the logical processor in the shallowest state, or the hardware C1 state. With the introduction of MWAIT, there is no longer a need for HLT. In modern systems, microcode converts HLT instructions into MWAIT instructions requesting hardware C1.

C-state Policy

In simple terms, the role of OSPM C-state policy is to predict the future. Shallow C-states are best used when the system has very short idle durations, for example, packet processing over a high-speed network. Deep C-states are best used when the system has long, uninterrupted idle durations; for example, idle time between submitting and completing an I/O. Figure 6-2 shows sample utilization from a single logical processor. It shows a mix of different shorter and longer idle durations, and ideal use of C-states.

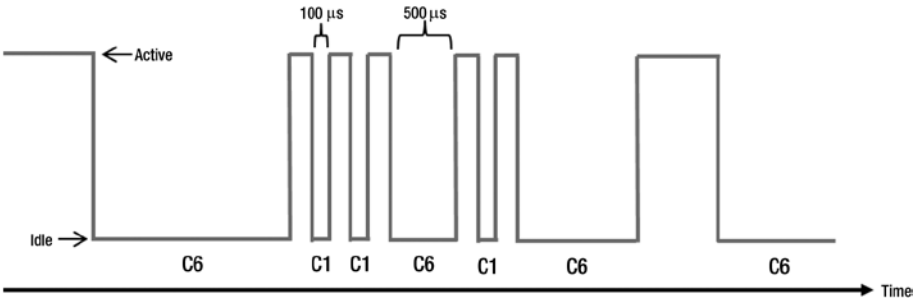

Figure 6-2. *Sample logical processor utilization with requested C-state*

With each logical processor generating unique utilization characteristics, OSPM policy needs to make individual decisions for each of them. Hardware is responsible for coordinating logical processor-level requests from two logical processors to determine the core C-state. For example, if one logical processor on a core is requesting C6 and the other is requesting C1, the core will go to C1. Similarly, hardware coordinates between core C-states to determine the package C-state. During hardware coordination, the shallowest of all C-state requests is used.

Optimal use of C-states is important for both power and performance. As discussed earlier, deep C-states introduce a performance penalty when they are exited. Using a deep C-state throughout a sequence of short idle periods results in a significant amount of stall time relative to the time spent executing instructions. In addition to latency penalties, there is also a transitional energy cost associated with C-state entry and exit. It may be counterintuitive, but using deep C-states throughout a sequence of short idle durations may actually result in higher power and lower performance. An idle duration has to be up to hundreds of microseconds long for the deepest C-states to save power—otherwise the energy cost of entry and exit has not been amortized.

C-state latency information exposed by ACPI is of limited use to OSPM C-state policy. It can be useful where software knows exactly what latency it can and can't tolerate. However, the latency value exposed by ACPI represents worst case latency, a value that can be several times greater than average or typical latency and is encountered only when in a package C-state. Cases where C-state latency is equal to the ACPI exposed value may never be encountered. In reality, there is no single value that represents C-state exit latency—it is variable and heavily dependent on other CPU activities. An additional downside of ACPI exposed latency is that it does not convey other performance visible effects such as cache and TLB flushes. These actions momentarily slow execution when the processor wakes up from a C-state due to running on caches that no longer contain recently used instructions and data.

C-state power consumption information exposed by ACPI also provides an incomplete picture from a power perspective. The power consumption of a logical processor, core, and package C-state are all very different, but ACPI objects expose only a single power value. In addition, ACPI C-state objects don't capture the idle duration necessary to break even from an energy perspective. In general, trying to convey to software accurate power consumption for some low-level state is problematic. Actual measured power depends on a number of external factors such as part to part CPU variation, temperature, and the efficiency power delivery components such as VRs and power supplies.

Chapter 2 discusses hardware C-state demotion mechanisms that processors use to prohibit deep C-states. If the PCU detects that such C-states are being used, the result may be a measurable performance impact or power increase. With these features in place, the OSPM C-state policies have the option of simply specifying the deepest state they tolerate, instead of trying to determine an ideal state, and allowing for further C-state selection optimization in hardware. This option has the benefit of eliminating complex software algorithms and simplifying the path to entering idle.

Processor Utilization

One metric OSPM C-state policy uses to determine whether to use deep or shallow C-states is *processor utilization*. This metric describes the percentage of time the processor is active or unhalted. The CPU provides fixed counters the OS can use to measure utilization on each logical processor. For example, the IA32_FIXED_CTR1 MSR described in Table 6-1 counts unhalted cycles at the rate of CPU base frequency (or P1 frequency) and the time stamp counter (TSC) MSR described in Table 6-2 counts all cycles, both halted and unhalted, at the same rate of CPU base frequency. Measuring the two events over some time period allows the operating system to calculate average utilization—the ratio of unhalted cycles in comparison to all possible cycles.

Table 6-1. *IA32_FIXED_CTR1 MSR (0x30B)*

Bits	Width	Field	Description
63:0	64 bits	Unhalted cycles	Always running measure of logical processor utilization. Increments when processor is active at the CPU base frequency of the part.

Table 6-2. *IA32_TSC MSR (0x10)*

Bits	Width	Field	Description
63:0	64 bits	Timestamp	Always running measure of logical processor time. Increments when the processor is active or idle at the CPU base frequency of the part.

In reality, a logical processor can only be in one of two utilization states at any given time, active (100%), or idle (0%). It's observing and averaging utilization of a logical processor over some longer time window that gives us the concept of being partially utilized.

Each OSPM policy has to decide over what length of time it will observe utilization. If the time window is too short, it's likely the average utilization result will be 0%, 100%, or something very close to those endpoints. If the time window is too long, the OS will fail to identify fleeting changes or other interesting characteristics in the variation of utilization. In addition to observing utilization over a single time window, OSPM may use multiple time windows to gain additional insight into longer term utilization trends. Another OS metric used for selection of C-states is known future activity, such as periodic timers that are due to expire soon, I/O outstanding, or repeating patterns of device interrupts. Unlike utilization, these methods rely on known events and can be predicted more accurately.

The complexity in OSPM accurately predicting idle durations, trends, or patterns in idle, and the potential for error in these predictions, does not prevent a good decision from being made. In fact, appropriate use of C-states can be realized with very simple C-state policies. In modern processors, deep C-state exit latencies are only a fraction of what they were when the technology was first introduced, so the impact of a suboptimal decision has been reduced significantly. Although there are environments with acute latency sensitivity, this is the exception rather than the norm. The majority of datacenter workloads see no measurable performance impact from the use of C-states.

P-state Control

P-states represent the most significant power performance tradeoff in a server today. In most systems, the power and performance impact of P-states is greater than the impact of all other power management features in the platform combined. The impact of OSPM P-state decisions is substantial. On a multi-socket system, a given decision could result in 100 watts lower power or cutting transaction response times in half. As such, OSPM P-state policies must identify a balance between low power and high performance that meets most of their user's needs. It's impossible to meet all users' requirements, so OSPM policies must also provide mechanisms for administrators to customize OSPM behavior to meet their needs.

Similar to C-states, software use of P-states has been greatly simplified over time. For example, early P-state implementations required the processor to be in a coordinated idle state, holding off system activity through the completion of a P-state transition. In modern processors, P-state transitions are low latency and are performed dynamically.

Software Controlled Interface

Software utilizes writes to MSRs rather than executing instructions to initiate P-state transitions. State transitions are initiated by OSPM writing a control value (or frequency ratio relative to base clock) to the IA32_PERF_CTL register. Table 6-3 describes this interface and its capabilities. The ACPI _PSS object discussed in Chapter 5 describes each individual P-state based on the control value. It acts as an identifier that hardware associates with an internal frequency and operating point.

Table 6-3. *IA32_PERF_CTL MSR (0x199)*

Bits	Width	Field	Description
7:0	8 bits	Reserved	Unused
15:8	8 bits	P-state target	Control value (frequency ratio relative to base clock) used to transition logical processor to the P-state target
31:16	16 bits	Reserved	Unused
33:32	1 bit	Turbo mode disable	Disables Turbo mode
63:34	31 bits	Reserved	Unused

IA32_PERF_CTL is accessible by the OS, by firmware, and by advanced applications with kernel-level privileges. There are cases where multiple entities may try to control P-states simultaneously. Although these cases are rare, where they exist, the administrator must take great care to ensure the capability is disabled for entities that should not have control.

The ACPI _PSD object (discussed in Chapter 5) informs the OS of its role and responsibility in coordinating P-state transitions. If all the logical processors in a CPU share the same P-state domain, it is not necessary for every logical processor to request a P-state change. OSPM policies have the ability to decide which of those logical processors should ultimately own the decision. Similarly, if all the logical processors in a CPU have their own unique P-state domain, the operating system must monitor each of them individually and make unique, logical, processor-specific requests.

Hardware provides several mechanisms OSPM can use to get feedback on policy decisions. The IA32_PERF_STATUS MSR provides a measure of the current frequency ratio, which conveys the specific processor P-state a core is utilizing. This mechanism is frequently used by software outside the OS to show the current operating conditions. Another mechanism for measuring frequency is the IA32_APERF and IA32_MPERF MSRs. IA32_APERF increments at the real frequency of a logical processor whereas IA32_MPERF increments at the CPU base frequency of the CPU. Measured over time, the ratio of IA32_APERF/IA32_MPERF allows OSPM to calculate the average frequency of a logical processor. Both IA32_APERF and IA32_MPERF can be reset by the operating system by writing them to zero, allowing OSPM to clear history from one observation window to the next. Some operating systems do not reset these registers, so IA32_APERF and IA32_MPERF can be used by other software outside the OS. Tables 6-4, 6-5, and 6-6 describe these mechanisms in greater detail.

Table 6-4. *IA32_APERF MSR (0xE8)*

Bits	Width	Field	Description
63:0	64 bits	Actual performance	Always running measure of time. Increments when processor is active at the current operating frequency of the part.

Table 6-5. *IA32_MPERF MSR (0xE7)*

Bits	Width	Field	Description
63:0	63:0	Measured performance	Always running measure of time. Increments when processor is active at the CPU base frequency of the part.

Table 6-6. *IA32_PERF_STATUS MSR (0x198)*

Bits	Width	Field	Description
7:0	8 bits	Reserved	Unused
14:8	7 bits	Core ratio	Frequency ratio of the core. This is multiplied by the base clock (typically 100 MHz) to get core frequency.
63:15	49 bits	Reserved	Unused

Collaborative Interface

A controversial topic that has been debated for years between hardware, firmware, and software engineers is, "Who is the ideal entity to control P-states?" Each of these entities has some unique information that provides insight to making an optimal P-state selection. For example, the OS has unique information about processes and threads, their priority, and dependencies between them. Hardware has unique insight into power, temperature, leakage, and resource scalability. Hardware can continuously monitor behavior and detect changes faster than an OS. Management firmware may understand a critical need to migrate a virtual machine (VM) off a server or have unique knowledge about response time requirements. A collaborative interface that allows control entities to specify requirements and hints, rather than discrete P-states, is a step toward a mutual decision.

Some of the latest CPUs contain support for hardware-controlled performance states (HWP), an interface for collaborative decision-making between hardware and software. HWP gives software the ability to supply a target frequency range to operate within along with performance guidance hints. This allows software to use its unique information to provide guidance and for hardware to optimize the selection of P-states within those software provided constraints. If software has no unique information to provide, hardware has the ability to autonomously select P-states.

There are a number of new capabilities this interface provides for control, feedback, and notifications that don't exist in the legacy interface. For example, this interface has the capability to indicate a change to guaranteed frequency initiated by an external power or thermal control policy. Tables 6-7 and 6-8 highlight the most important interface capabilities—the primary ones used by OSPM to cooperatively manage P-states. HWP MSRs are documented in detail in the Intel Architecture Software Developer Manuals.

Table 6-7. *IA32_HWP_CAPABILITIES MSR (0x771)*

Field	Description
Highest performance	Value for the maximum non-guaranteed performance level (aka Turbo).
Guaranteed performance	Current value for the guaranteed performance level. Can change dynamically as a result of internal or external constraints (e.g., thermal or power limits).
Most efficient performance	Value of the most efficient performance level (aka P-state with the lowest voltage).
Lowest performance	Value of the lowest performance level.
Reserved	Unused.

Table 6-8. *IA32_HWP_REQUEST MSR (0x774)*

Field	Description
Minimum performance	Minimum performance allowed in P-state selection.
Maximum performance	Maximum performance allowed in P-state selection.
Desired performance	A performance hint to hardware within the performance range defined by IA32_HWP_CAPABILTIES (see Table 6-7). If set to zero, hardware makes this decision autonomously.
Energy performance preference	A performance hint to hardware that influences the rate of performance increase/decrease and the result of hardware optimizations.
Reserved	Optional or unused.

Firmware Control

The previous sections detailed some of the challenges software faces in the selection of P-states, such as how to strike a balance between low power and high performance that will meet most users' needs. Another challenge software faces is ensuring that new power management features work correctly on older versions of software.

This is a commonly faced challenge as new hardware is frequently coupled with old software. For example, a server utilizing hardware released in 2014 may be running software released in 2009. *Forklift upgrades*, where software and hardware are simultaneously upgraded, are uncommon due to cost, complexity, and integration risk. In many datacenters, hardware platforms are upgraded or replaced several times under the same software stack. This mismatch between the introduction of new hardware and new software creates an environment where servers may not be fully enabled or optimized for the latest power management features. This is very different from client products that can rely on new hardware being coupled with new software as well as some product-specific device drivers for power and thermal management. Figure 6-3 illustrates the differences in energy efficiency between different Linux kernels without changing the server, applications, or workload. Between 2008 and 2012, idle power decreased by 35 W and active power at a given throughput level decreased by up to 15 W.

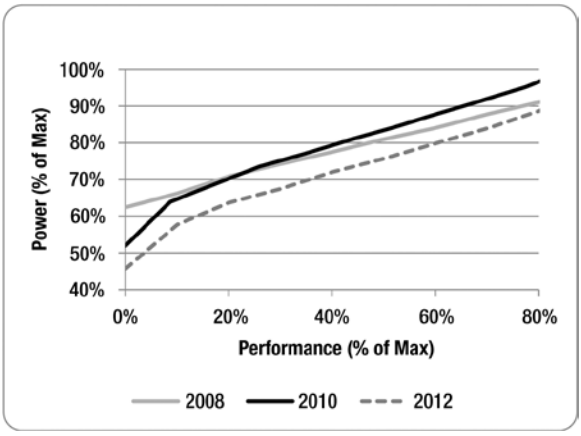

Figure 6-3. *Power impact of different Linux kernels*

Another challenge in a datacenter is enforcing a consistent set of power and performance tradeoffs across systems running different operating systems. Each OSPM policy makes its own unique choices about power state selection leading to unique behaviors. One may be biased toward maximizing performance while another may be biased toward minimizing power. There are cases where it is desirable to have a single OS-agnostic power management policy that can be applied across many different software environments.

It is not practical for an administrator to continually upgrade software to enable the latest hardware feature support and optimizations. Similarly, it is not practical for an administrator to fine-tune one OS to behave like another. To address these issues, several

server manufacturers provide their own P-state control capability that is implemented in firmware. This allows an administrator to apply a uniform P-state control policy to an entire rack of servers that may be using a variety of different operating systems and versions. Similar to many OSPM implementations, these firmware policies allow administrators to specify a preference toward low power, high performance, or a balanced setting. These types of advanced configuration options are available as BIOS setup options, allowing administrators to select between firmware or software-controlled policies.

P-state Policy

P-state policy is responsible for adjusting the performance capability of the processor to meet current or expected demand. When logical processor utilization is low, demand can be met by running the processor at low frequency. As logical processor utilization increases, frequency must also increase to prevent the system from reaching full utilization.

Similar to C-states, P-state policy is based primarily on logical processor utilization. Individual policies may supplement utilization with additional information such as the current operating frequency, or the priority of processes running. Utilization calculation and time window considerations discussed for C-states are similar for P-states including the use of IA32_PERF_CTR1 and IA32_TSC. Similar to C-states, the OS makes individual decisions on behalf of each logical processor where the ACPI P-state domain indicates this is necessary.

Performance Capacity

There are two ways a server can increase throughput, or the number of instructions executed per second. The first is to utilize available resources. This is as simple as utilizing idle cycles on a processor whenever that processor is not fully utilized. The second is to increase the speed or rate at which a logical processor executes instructions. This is accomplished through increasing frequency.

There are many cases where processor utilization alone is not sufficient to understand the operating conditions of a processor. For example, if a processor is 50% utilized running at 1.0 GHz (its minimum frequency), that processor has significantly more performance headroom than if it was 50% utilized running at 3.0 GHz (its maximum frequency). Although processor utilization is the same in both cases, the operating conditions are very different.

An important concept to introduce that aids in the understanding of processor operating conditions and OSPM P-state policies is performance *capacity*. This metric is based on processor utilization, but it takes into account the impact of running at a higher or lower frequency to accurately capture performance headroom. The capacity metric represents the amount of the maximum guaranteed throughput capability the processor is currently using.

$$Capacity\,(\%) = Utilization\ at\ CPU\ Base\ Frequency * \frac{Current\ Frequency}{CPU\ Base\ Frequency}$$

A capacity of 100% might represent a processor running at 100% utilization while at the CPU base frequency. This is the guaranteed throughput capability of the processor. With Turbo, which is a non-guaranteed frequency, it is possible for capacity to exceed 100%.

Figure 6-4 illustrates how OSPM uses frequency to achieve maximum throughput. Up to 40% of maximum throughput is achieved at the lowest frequency, simply by utilizing idle cycles. At the point at which logical processor utilization starts to approach 100%, OSPM P-state policy increases frequency to provide additional throughput. When the processor reaches 90% of maximum throughput, it is running at the highest frequency, and the last 10% of idle cycles are utilized to reach maximum throughput. This chart also serves as an illustration of a P-state policy designed for best energy efficiency.

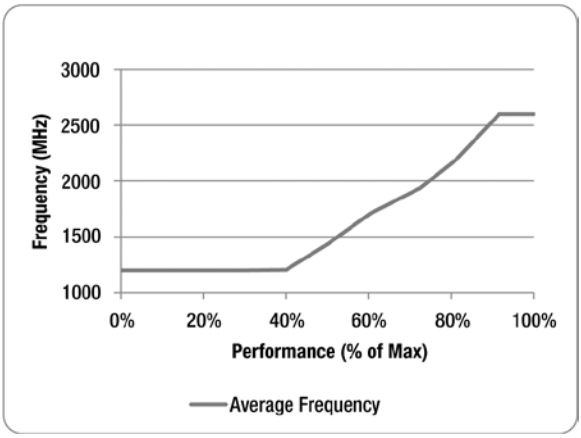

Figure 6-4. *Comparison of frequency across a full range of performance*

From a capacity standpoint, a 100% utilized processor running at 1.5 GHz and a 50% utilized processor running at 3.0 GHz are identical. Running at high utilization and low frequency provides lower power and higher response times whereas running at low utilization and high frequency provides higher power and lower response times. The performance impact of P-states is realized in a datacenter in terms of higher or lower response times.

Where there is sustained load on the system, a P-state policy should never impact maximum throughput. Use of P-states varies between operating systems, and operation at a given capacity can be accomplished in many ways. Whether to meet an increase in compute demand by running at higher utilization or by running at higher frequency is a key decision OSPM P-state policy needs to manage. Figure 6-5 shows a comparison between average logical processor capacity and average logical processor utilization.

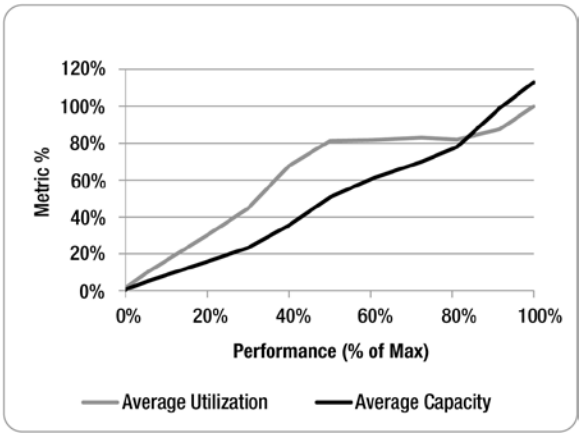

Figure 6-5. *Comparison between average utilization and average capacity*

■ **Note** Datacenter administrators typically describe server utilization in terms of logical processor utilization, or what is displayed by monitoring and profiling tools. Describing utilization without taking into account frequency is terribly misleading since a server running at 40% utilization may in fact only be running at 25% capacity.

In Figure 6-6, capacity exceeds 100% because the metric is based on the CPU base frequency, or the guaranteed frequency of the processor. As discussed earlier, Turbo mode is a non-guaranteed frequency, so the chart illustrates performance due to Turbo being above and beyond the guaranteed performance capacity of the processor. Capacity isn't a perfect metric either; it assumes exceptional frequency scaling (ratio of the percentage increase in performance to the percentage increase in frequency). In reality, frequency scaling varies from system to system since it is heavily dependent on the performance of the cache and memory subsystem and is influenced by unique workload and system characteristics.

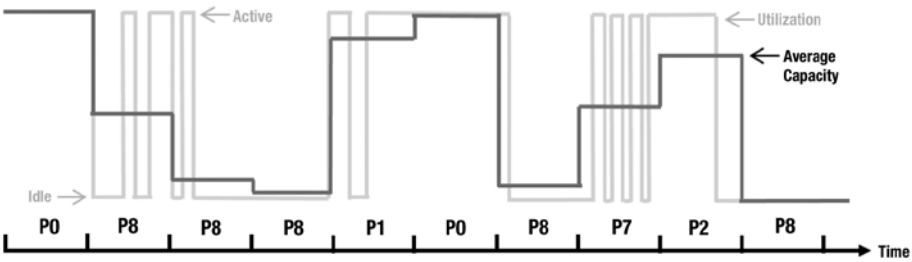

Figure 6-6. *Sample logical processor utilization with requested P-state over one second*

Figure 6-6 shows sample capacity from a single logical processor and energy efficient use of P-states over one second of time. For simplicity, the example assumes nine P-states with the lowest frequency mode at P8, the CPU base frequency of the processor at P1, and Turbo at P0. During periods of light activity, P8 is frequently selected, even when the capacity is greater than 30%. Nearly half of the throughput capability of the processor can be satisfied by utilizing idle cycles at P8 alone.

There are cases outside of Turbo where the CPU will run logical processors at a higher frequency than the OS requests. The integration of memory controllers and I/O (PCIe) in the processor creates cases where logical processor utilization alone is insufficient to determine if frequency is limiting performance. For example, it is possible for network intensive workloads to drive line-rate traffic at very low processor utilization. Due to this low processor utilization, a typical OSPM policy will select a low frequency P-state. However, in this case, running logical processors at the highest frequency will increase throughput through decreasing latency. With OSPM lacking visibility into these types of bottlenecks, modern processors include special features to detect and handle these cases. The CPU will autonomously increase processor frequency in cases of high IO bandwidth to ensure maximum throughput is not impacted.

P-state Coordination

With P-states, as is the case for T-states and C-states, it is possible to change the way coordination happens between software and hardware. Specified via ACPI, power state coordination can use one of the following methods. (The vast majority of server operating systems utilize HW_ALL.)

- **HW_ALL:** The OS makes state transition requests for each logical processor and treats them as independent. The OS does not need to consider any effects on other logical processors. Hardware takes care of any coordination needed. Where there are two logical processors in the same core requesting P1, and one of the logical processors changes its request to P*n*, the core remains at P1.
- **SW_ANY:** The OS makes a state transition request for a single logical processor in the same domain and the effect is immediate and independent of previous requests. Where there are two logical processors in the same core requesting P1, and one of the logical processors requests P*n*, the core goes to P*n*.
- **SW_ALL:** The OS makes state transition requests for each logical processor in the same domain and treats them as dependent. For example, if there are sixteen logical processors in a single domain, the OS changes P-state for that domain by making the same request on each of the sixteen logical processors.

■ **Note** While P-state coordination is controlled by BIOS firmware, selecting a mode other than HW_ALL can result in undesirable behavior because most operating systems have not been enabled or optimized for other modes.

T-state Control

Operating systems also have the capability to control T-states, or throttling states, described in Chapter 2. Modern server processors all include the appropriate temperature sensors and hardware mechanisms to prevent the processor from overheating, so OS control of T-states is no longer necessary. Using T-states to improve energy efficiency is generally not done nor is it recommended. Unlike P-states, voltage is not scaled for T-states—using T-states results in substantial decreases in performance for only small decreases in power.

Global Power Policy

There are cases where the default C-state or P-state policy set by the OS may be out of sync with user performance requirements. Operating systems have two different mechanisms for tuning power management to address this. The most common is support for predefined power policies. Operating systems provide options for the administrator to set a global power management policy that will change the behavior and selection of C-states and P-states to be more biased toward low power, high performance, or a balanced approach. For example, the Windows Server control panel exposes options such as high performance, balanced, and power saver that change the way the internal OSPM algorithms select C-states and P-states. A second type of tuning is possible using advanced settings or options accessed using low-level tools or interfaces that allow administrators to customize their own policy.

As discussed in Chapter 2, there are a variety of other power management features outside of C-states and P-states that impact power and performance. Most of these power states across memory, caches, and processor interconnects are not explicitly controlled by OSPM. The visibility required to make power state decisions only exists in hardware. Although OSPM has no low-level control over these power features, hardware does expose a mechanism for the OS to specify an energy policy preference.

Energy policy preference, or OSPM bias toward low power or high performance, is controlled by the IA32_ENERGY_PERF_BIAS MSR described in Table 6-9. The register is used to define the desired balance between power and performance for those power management features that are not explicitly controlled by OSPM. In environments with strict response time requirements, it's preferable to use the minimum IA32_ENERGY_PERF_BIAS value rather than disabling features, since disabling power management features can increase power and temperature, ultimately impacting maximum performance. IA32_ENERGY_PERF_BIAS is dynamic so it can be changed at runtime as OSPM sees fit.

Table 6-9. *IA32_ENERGY_PERF_BIAS MSR (0x1B0)*

Bits	Width	Field	Description
3:0	4 bits	Energy policy preference hint	Represents a sliding scale where the value 0 is maximum performance and the value 15 is minimum energy. A value of 7 roughly translates to a balanced policy.
63:15	49 bits	Reserved	Unused

Process Scheduling

The OS process scheduler doesn't directly interact with hardware power management features, but its scheduling policies can have a significant effect on energy efficiency. Many scheduler features that improve performance, such as starvation prevention (ensuring every process gets a chance to run), unique treatment of CPU and I/O bound processes, load balancing, and migration optimizations, have a positive impact on energy efficiency. However, there are many features where performance benefit has a negative impact. In addition, the answer to the question of whether or not various scheduler features improve energy efficiency can depend on the target microarchitecture. Some of the key factors that impact kernel scheduler energy efficiency include awareness of logical processor, core, and package topology and capabilities, timer tick frequency, and execution consolidation.

Topology and Capability Awareness

A key piece of information an OS scheduler needs in order to make energy-efficient decisions is to what degree multiple hardware threads or logical processors on the same core share resources. Where there is very little resource sharing between logical processors, a process scheduler can treat them as independent execution resources. However, where there is substantial resource sharing between logical processors, a process scheduler needs to understand how the throughput capability of an individual logical processor is impacted as additional logical processors sharing the same resources are concurrently utilized.

With a large number of servers running at low utilization, the process scheduler has a key decision to make in terms of how to utilize logical processors. One strategy is to utilize a single logical processor on every core before utilizing any of the simultaneous multi-threading (SMT) "sibling" logical processors. This has the benefit of giving logical processors dedicated access to otherwise shared core resources, keeping logical processor utilization low. Low logical processor utilization leads OSPM to use lower power P-states. Another strategy is to utilize both logical processors on a core before utilizing any of the additional cores. This limits coherency traffic and minimizes resource use, improving C-state residency. It may allow some of the cores to remain in long uninterrupted idle durations.

There is no single right answer for the best strategy to pursue since the optimal decision is microarchitecture specific. Your strategy depends on the amount of resources shared between logical processors; it is influenced by cache sizes and levels of the cache hierarchy; and it is subject to the different types of C-states available and the latency to transition in and out of them. To some extent, it even depends on unique workload characteristics. Generally speaking, the majority of server processors benefit from spreading software threads out over as many cores as possible. Typically, the scheduling decision that keeps core frequency lowest is the most energy efficient. When a scheduler utilizes all logical processors on the same core before scheduling on the next cores, resource contention increases utilization and leads to OSPM use of higher power P-states earlier. It takes a substantial increase in C-state residency to offset even a single step increase in P-states.

Another strategy from a process scheduling standpoint is to utilize all the logical processors on one CPU before utilizing logical processors on the other CPU. Conceptually, this makes a lot of sense as a CPU executing no instructions should be able to enter very low power states. In reality, benefits with this approach are mixed. In a multi-socket system with two or more CPUs, any single CPU cannot independently go to a deep package C-state while any other CPU in the system is active. There can be a significant amount of memory and I/O device activity on a CPU even when all its logical processors are idle due to the integration of the memory controller and PCIe.

Another challenge with utilizing only logical processors on one CPU is the impact to the non-uniform memory access (NUMA) locality. If remote memory references increase as a result of a scheduling decision, it can increase response times and decrease energy efficiency.

OSPM P-state policies that are optimized for performance may see an energy efficiency benefit to this approach as they use high-frequency P-states even at low utilization. In these environments, utilizing all the logical processors on one CPU before utilizing logical processors on the other CPU limits the number of CPUs running at high voltage.

Timer Tick Frequency

A periodic timer interrupt determines the frequency at which the operating system will perform necessary scheduling tasks. This periodic timer or timer tick plays a significant role in energy efficiency, both when the processor is active and when it's idle. When idle, logical processors must wake up to handle timer ticks, interrupting the coordination necessary to enter a deep package C-state. Modern operating systems suppress the timer tick or limit the number of timer ticks when the system is idle. For example, at idle, timer ticks may be received by a single logical processor that is responsible for forwarding the interrupt only to active logical processors. This improves the average idle duration for logical processors and results in improved package C-state residency. This action is commonly called tick skipping, dynamic ticks, or tickless idle.

When the processor is partially utilized, high-frequency timer ticks can lead to higher performance, but this comes at a power cost. A kernel with a 1000 Hz tick rate delivers a timer interrupt every 1 millisecond whereas a kernel with a 64 Hz tick rate delivers a timer tick every 15.6 milliseconds. Linux has the option to get rid of timer ticks even when busy.

From a performance perspective, running with a higher frequency tick results in lower response time. The increased time accuracy from a higher frequency tick improves the resolution of timed events—it leads to more precise process preemption and it improves enforcement of priority and fairness policies. For example, a high-priority process that is reading data from a drive will have less latency between drive reads with a high-frequency timer tick. From a power perspective, systems running with a higher frequency timer spend more time handling timer interrupts. This increase in interrupt handling time drives logical processor utilization higher and splits long idle durations needed for deeper C-state entry into several shorter idle durations. Increased interrupt rates also increase the number of C-state transitions, accumulating more C-state exit latency and adding more transitional energy.

Execution Consolidation (Core Parking)

A process scheduler spreads software threads across as many logical processors as possible, avoiding scheduling multiple software threads on the same logical processor until there are no more free logical processors available. This maximizes use of parallel resources and minimizes process response time. There are some cases from a performance, power, or thermal perspective when it is preferable to do the opposite—scheduling multiple threads on a single logical processor to leave other logical processors idle. When execution is consolidated in this manner, the OS must also ensure the handling of device interrupts and other timed events is done only on the subset of active logical processors.

Energy Efficiency

Scheduling software threads to maximize utilization on a single logical processor while leaving other logical processors idle increases deep C-state residency and minimizes power state transitions. Execution consolidation has many different names in products and research such as core parking, core idling, and power-aware scheduling. Creating an imbalanced load in this manner keeps some number of logical processors in an uninterrupted idle state.

Although execution consolidation does increase deep C-state residency, it doesn't necessarily improve energy efficiency—whether or not this practice benefits energy efficiency is dependent on many factors. One key consideration is the amount of power that comes from the cores in comparison to the remaining uncore components, such as the last level cache, memory controllers, and processor interconnects. Execution consolidation is not favorable for CPUs that have a significant amount of power coming from uncore components. It takes only a single active logical processor to keep the uncore components in an active state. In these cases, the percentage increase in power from running a workload on two cores compared to one core is very small. Execution consolidation is more favorable where a CPU has non-core resources, such as mid-level caches, shared by small subsets of cores, but not all cores. The more that cores and resources shared by small subsets of cores account for total CPU power, the more favorable execution consolidation becomes.

Execution consolidation can increase response times as only one software thread can run at any given time on a logical processor. With execution consolidation, software threads that are ready to execute may need to wait to execute to avoid utilizing those

logical processors kept in an uninterrupted idle state. This practice typically results in active cores running a high utilization, also limiting the time a CPU can utilize package C-state states. Another consideration is the degree to which the software threads share data. Where there is substantial data sharing, particularly data that is frequently modified, execution consolidation can improve cache locality significantly. This characteristic is not clearly visible to a process scheduler, so dynamically identifying beneficial cases is difficult. Where cache benefits are realized, the reduction in execution time and decrease in interconnect and memory utilization can offset some or all of the cost of increased execution time. The same execution consolidation concept can extend to CPUs in multi-socket systems. However, it is common for the memory controller and I/O (PCIe) in the CPU to be active even if no software threads are running. This uncore activity consumes a significant amount of power. As a result, simply moving software threads off of a socket will not allow that socket to enter a low power idle state.

In measuring energy efficiency, a key consideration is how execution consolidation is used in conjunction with P-states. For most processors, actions taken by the process scheduler that cause increases in logical processor utilization typically lead to increased use of higher frequency P-states. In these cases, energy efficiency suffers because even very small increases in core voltage result in very large increases in power. In practice, it takes very specific or synthetic workloads on very specific processors to show an energy efficiency benefit from execution consolidation, but these cases are the minority.

Power Capping

A more universally beneficial application of execution consolidation is for *power capping*, that is, managing the system so it is always operating below a defined power limit. The primary mechanism for server power limiting is by placing hard limits on logical processor frequency and memory bandwidth. When these mechanisms are exhausted and the power limit is still not met, platform firmware can initiate logical processor idling to reduce power further. In cases where there is a failure in power or cooling infrastructure, it may take several different power limiting mechanisms to meet the power limit and avoid system shutdown.

ACPI includes an optional processor aggregator device that provides an interface and control point for firmware to idle logical processors. When additional power limiting is necessary, firmware requests a specific number of logical processors using the ACPI _PUR method, and the OS satisfies this request.

There is no energy efficiency benefit to power limiting when the practice is characterized on a single server. However, in cases where a rack of servers are operating under a rack-level power limit, datacenter management software can limit power of less critical servers in order to give additional power budget to more critical ones.

Single-Threaded Performance

Another application of execution consolidation is to improve single-threaded performance. As discussed in Chapter 2, Turbo frequency increases with the number of idle cores in the CPU. Restricting the key software thread along with device interrupt handling, timed events, and any other background activity all to a single core ensures the active core is always running at the maximum Turbo frequency.

Memory Management

Similar to the concept of execution consolidation, if a server is using less than its full memory capacity, it is possible to consolidate memory references to a subset of memory. This allows remaining memory to enter uninterrupted low-power states. There has been significant investigation in this area, but only a limited number of scenarios realize a benefit.

Memory consolidation requires the OS to understand the physical memory topology. Similar to CPU topology, this information is conveyed to the OS through ACPI. The Memory Power State Table (MPST) describes physical memory in terms of memory power nodes, or specific address ranges that are power managed as a single entity. There can be several address ranges within a single node because these ranges may not necessarily be contiguous. In addition, MPST describes the power states supported by hardware including power consumption and exit latencies. Similar to OSPM for C-states and P-states, this level of information is critical for the memory manager, so it has the right inputs to determine power saving and performance impact.

Unlike C-states and P-states, the OS is not required to initiate power state transitions for memory nodes. Hardware autonomously transitions power nodes between appropriate power states based on their activity level. Memory nodes are typically defined at the channel level instead of a rank or DIMM level. The reason for this is because the lowest power states for memory, such as self-refresh, are applied at the channel level. Figure 6-7 shows the relationship between physical memory and memory power nodes. A memory power node could include from one to three DIMMs.

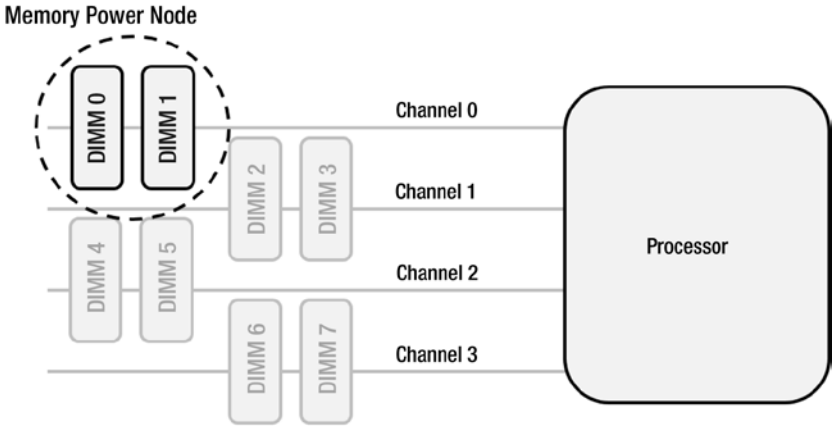

Figure 6-7. *Relationship between physical memory and memory power nodes*

Equipped with the physical memory topology, latency, and power information, an OS memory manager can make power-aware decisions about memory allocation. First, physical memory can be allocated so that DIMMs are used to capacity on one channel before DIMMs are used from remaining channels. Next, the OS memory manager can periodically relocate physical memory. This is necessary as a large amount of unused physical memory is uncommon and memory becomes fragmented over time. Relocation puts frequently referenced memory into one subset of power nodes, and infrequently referenced or unallocated memory into another subset of power nodes. Although this practice doesn't ensure that one subset of nodes is always idle, it does allow for significant residency improvements in the deepest power states.

As is the case with other power management features described in this chapter, memory consolidation needs to balance the need for low power with the need for high performance. The default settings for a server are to interleave memory across channels and across ranks of memory. This increases the bandwidth capability of the system by spreading large regions of allocated memory evenly across channels, ranks, and DIMMs. In order to enable power-aware memory management, this interleaving needs to be disabled by BIOS firmware when the memory controllers are initialized. From a memory bandwidth perspective, many datacenter workloads use only a fraction of the bandwidth capability, so no performance impact is realized. For other workloads with higher memory bandwidth requirements, this is an unacceptable trade off. It is possible for an operating system to maintain some level of memory interleaving to accommodate bandwidth demands, but the overhead of managing this interleaving in software would eliminate any of the potential benefits.

Another challenge facing memory reference consolidation is the complexity of relocating memory. Relocating memory adds overhead in terms of additional processor time and additional memory traffic, both of which reduce the time spent in low-power states. Workloads that change their memory reference characteristics over time or memory pages that are repurposed over time make it difficult to predict what pages will be hot or cold. Finally, not all regions of memory can be relocated, such as reserved or non-paged memory.

The potential benefit of memory consolidation is being reduced over time as datacenters transition from DDR3 1.5 V to DDR3L 1.35 V to DDR4 1.2 V. Not only is active memory power lower, but the power differences between shallow memory power states applied at a rank level and deep memory power states applied at a channel level are decreasing. The complexity of power-aware memory management leads to limited applicability. It requires very specific workloads and system configurations to show significant energy efficiency benefits.

Device Drivers

In addition to processor drivers that implement C-state and P-state control policies, some I/O device drivers are also responsible for implementing device power management and control policies. Other devices autonomously manage power and do not define software interfaces for power state discovery and control. Most servers can't tolerate the latency of deep D-states in a production environment. In cases where the latency impact is negligible, some devices monitor their own activity and utilize shallow states. In other cases, many D-states features are unused or disabled on servers.

PCIe, SATA, and USB

As discussed earlier in the chapter, operating systems use PCI-SIG defined standardized interfaces for discovering PCIe D-state capabilities, for putting devices into a D-state, and for querying the power status of devices. The OS PCI driver discovers the supported capabilities from PCI configuration space when it enumerates the PCI devices. D-states for PCIe, SATA, and USB devices are managed through similar native hardware interfaces. ACPI is used to augment or supplement these capabilities' for example, to handle devices that lack native hardware interfaces, such as end-point devices on I2C (standard bus for attaching low-speed peripherals). Similar to C-states and P-states, software coordinates with hardware to initiate D-state transitions.

Software-initiated D-states require a greater level of coordination than do processor power states because they involve device, host, bus, and class drivers. Device drivers are responsible for saving and restoring device context before and after devices are put into low-power states. In addition, the OS has the additional responsibility of ensuring devices have no pending transactions before initiating entry into a D-state. Additionally, the OS must ensure that all devices on a bus are in a low-power state before putting a bus in a low-power state. Due to the latency of device power management, server operating systems typically initiate D-states based on entry into a higher level system state, such as S3. Prior to entering S3, the OS is required to put each device into a D3 state. The OS maintains specific mappings between S-states and different D-states bus and device components must go into prior to entering a target S-state.

Graphics

Modern server processors may also include integrated graphics processors. One of the more complex cases of I/O device driver power management is for graphics because, in addition to the responsibility of managing D-states, the graphics driver must also manage graphics processor P-states. Similar to a processor driver, the graphics driver's P-state request is also based on events that measure the current level of activity. Device drivers use a combination of demand, latency, and frame per second (fps) requirements in determining the appropriate P-state. The graphics driver updates the PCU with information in addition to the required core and ring performance to keep graphics running effectively.

With integrated graphics, processor cores and graphics share the same package power and thermal budget. This is unlike most servers, which feature a discrete graphics controller. With integrated graphics, it's not possible for both graphics and processing cores to be active in the highest frequency state at the same time. Two different device drivers making requests for higher performance and power states on the same CPU without coordination between them can cause issues where either the processor cores or graphics aren't getting their requested performance. When the power budget cannot satisfy the requests of both drivers, the PCU makes decisions based on its own internal knowledge to best balance the power budget.

Graphics devices also utilize C-states, but these are controlled autonomously in hardware. Graphics hardware detects when resources are idle and handles C-state entry including context save and restore. C-states are immediately exited whenever graphics hardware is accessed by the device driver.

Virtualization

A virtual machine monitor (VMM) has all the same responsibilities for power management that a native operating system does. The VMM, or host, is responsible for controlling C-states, P-states, and global power policy. In addition to these mechanisms, VMMs enable several additional capabilities that improve energy efficiency such as server consolidation or new approaches to power management enabled by virtual machine (VM) migration.

Power State Control

VMM management of C-states, P-states, and global power policy is not a responsibility shared with guest VMs. These features are host-controlled. If guest VMs were allowed to access power state control capabilities, it would create a number of policy conflicts between concurrently running VMs. Some VMs, with no knowledge of other VMs, would require high performance, biasing state selection toward shallow idle states and high-power active states. Some VMs would require low performance, biasing state selection toward deep idle states and low-power active states, and other VMs would request everything else in-between.

Allowing the host to control power management and limiting or eliminating the role of guest VMs is common practice across the various models for virtualization, such as the hypervisor model used in ESXi, the host-based model used by Hyper-V and the kernel-based virtual machine, or KVM, and the hybrid model used by Xen. Each of these VMMs boots VMs using virtual BIOS. The virtual BIOS exposed to guest VMs has a different set of capabilities than the physical server's own BIOS firmware. To facilitate host-controlled power management, the virtual BIOS does not expose power management features to the guest VM. This prevents guest VMs from unnecessarily loading drivers and control policy for C-states and P-states—all guest VMs need to do is execute a halt. This eliminates additional overhead introduced by guests making state requests that would be ultimately ignored by the host.

There are some cases where guests are *enlightened*, or aware they are running in a virtualized environment. These guests may be given certain control capabilities or may have an awareness of power management features that is not typical of guest VMs. This enlightenment allows for some power management optimizations by the host. For example, different C-states or S-states may be exposed to guests, allowing individual VM's power state selections to act as feedback for the host. These requests allow a host power management policy to consider individual VM requests along with its own policy to make optimal system-level decisions.

Idle Considerations

Virtualized environments present some unique challenges to getting to low idle power. Even an operating system doing nothing generates significant activity when there are many different guest VM operating systems loaded. Similar to native environments, or environments that run a single operating system, some minimal amount of activity exists at idle, such as periodic timer interrupts or network heartbeats. These periodic events continually wake up logical processors and can prevent the system from entering deep

package C-states. Even a minimal amount of activity becomes significant when it is multiplied by a large number of VMs on a single physical system.

Similar to native operating systems, many VMM have optimized idle scenarios in an attempt to align periodic events and reduce the number of times logical processors are woken up. For example, timer ticks for different VMs (especially those sharing a common frequency), can be aligned to happen at the same time. This causes periodic event handling for multiple VMs to happen in parallel, reducing the number of power state transitions and increasing average idle residency.

Figure 6-8 compares VMM and guest idle scenarios across several years of software improvements. In the figure, the base system idle power is approximately 100 watts. The 2009 software stack uses a combination of guest operating systems released at that time and earlier, with varying degrees of idle activity optimization. The VMM in this case is doing little to no periodic activity alignment. The 2014 software stack uses a combination of guest operating systems released at that time and earlier and the VMM aligns periodic activity. This comparison illustrates the importance of software components in achieving low idle power in a virtualized environment.

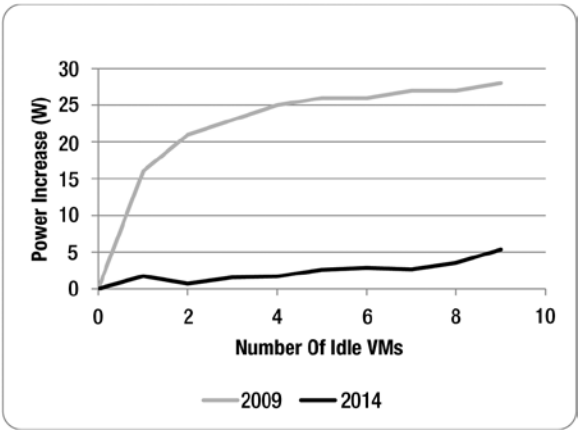

Figure 6-8. *Idle power impact due to software in a virtualized environment*

Active Considerations

Utilization characteristics in a virtualized environment are typically different than a native environment. Guest VMs are limited in the number of logical processors, the amount of memory, and the amount of network bandwidth they can utilize. These limits vary based on instance type and VM size. Restricting applications to different subsets of system resources leads to asymmetric resource utilization. Resource utilization in a native environment is typically very similar across processors and sockets because applications have the ability to utilize all available system resources. In a virtualized environment, it's not uncommon to see some logical processors running at sustained 100% utilization, while others are idle or at low utilization. These significant differences in utilization characteristics affect the OSPM policy's ability to make P-state decisions that accommodate a wide variety of VM performance requirements.

Figures 6-9 and 6-10 illustrate typical logical processor utilization at a fixed throughput rate. In the example of a native environment, the system experiences very few changes in utilization characteristics and many of the logical processors have similar utilization levels. In the example of a virtualized environment, the system experiences frequent changes in utilization characteristics with great variation in utilization across logical processors. These qualities lead to a challenging set of decisions for OSPM. In order to meet the performance requirements of a wide variety of VMs, virtualized environments are typically much less aggressive in their use of power management.

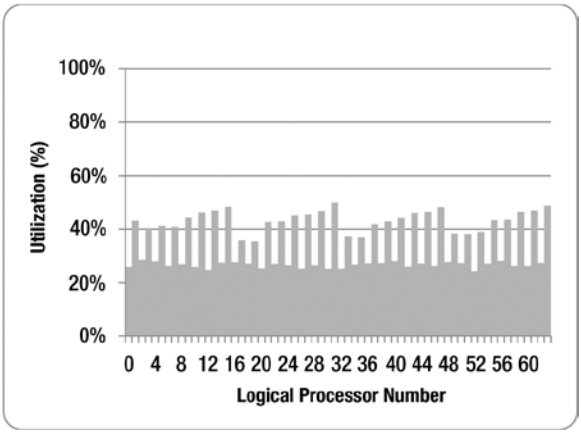

Figure 6-9. *Example of 30% throughput in a native environment*

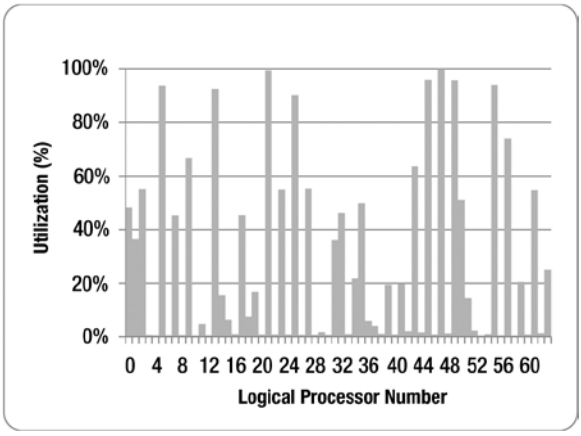

Figure 6-10. *Example of 30% throughput in a virtualized environment*

Another consideration from an energy efficiency standpoint is the overhead of additional software layers present in a virtualized environment. This overhead can increase execution or response time and decrease throughput. Either of these impacts increases power as transactions or computational tasks take longer to complete. The impact of virtualization overhead increases as more virtual servers are consolidated onto the same physical server. CPU bound workloads tend to experience less impact from overhead, typically less than 10%, since there is little to no kernel time. I/O-bound workloads that execute lots of system calls, retire a lot of privileged instructions, and generate a lot of interrupts typically see greater than 15% overhead.

As a result, software or hardware enhancements that limit or eliminate the overhead of a VMM provide significant improvements in energy efficiency. Virtualization technologies such as EPTs (extended page tables) or VT-d (virtualization technology for direct device assignment) are typically thought of as performance optimizations, but running workloads on systems with and without these features enabled demonstrate their capability as a power optimization.

Consolidation

The majority of native servers, or servers that run a single operating system, support only one primary application. Limiting the number of applications has several benefits in this scenario. It improves performance, it avoids resource conflicts, it improves the ability to monitor applications, and it encapsulates problems. In many cases, running a single application on a server also leads to low CPU utilization.

The single-best method for improving energy efficiency of a server is to increase utilization. The energy cost of a server transaction is inversely proportional to utilization. For example, the average energy cost of a server transaction is up to five times lower at 60% capacity than it is at 10% capacity. Server consolidation is one of the primary mechanisms for increasing utilization. It allows a number of existing servers, such as servers running at low utilization or servers running with low-performance processors, to be replaced by a single higher performance server. The operating systems and applications from existing servers are consolidated and deployed as guest VMs running under a VMM on the higher performance server. Even though multiple applications are running on the same physical server, these applications continue to gain many of the same isolation benefits realized with a 1:1 mapping between applications and physical servers.

The benefits of consolidation from an energy efficiency perspective are immense. The primary benefit is the power savings realized by decreasing the number of physical servers required to support the same set of applications. Most virtualized environments have greater than a 10:1 consolidation ratio (the number of virtual servers running on a single physical server), so the opportunity to save power is tremendous. After server consolidation, many of the older and higher powered servers can be powered off and removed completely. The next benefit of consolidation is the ability to increase the average utilization of servers. Physical servers running a VMM supporting multiple applications typically have much higher utilization than native servers supporting a single application.

■ **Note** Consolidation is often limited by non-CPU constraints such as the amount of DRAM in the platform or performance requirements defined in service level agreements.

Servers running legacy software, or older applications and operating systems, are frequent targets for consolidation. Older operating systems may not have optimal support for the latest hardware power management features or may have some power management features by default. For example, environments such as Windows Server 2003 or Linux distributions using pre-2.6.5 kernels do not enable the most beneficial power management features out of the box. In this context, consolidation provides additional benefit by enabling legacy software environments to use the latest OSPM policies. When guests with legacy software are run under a modern VMM, many of the latest software power management benefits are realized.

VM Migration

A server running at 25% of maximum performance is energy proportional if it consumes no more than 25% of maximum power. Technologies such as C-states, P-states, and memory CKE (Clock Enable) are key ingredients in the pursuit of energy proportionality; however, servers still have a long way to go to meet this goal.

Migration allows VMMs to move a guest VM from one physical server to another without service interruption or loss of execution context. There are many reasons to move a VM from one physical system to another. The most common reasons are to perform maintenance on a system or to vacate a system that is experiencing some type of performance or functional issue. Datacenter management software also uses VM migration to load balance virtual servers across physical servers. Using migration and consolidation together, some servers can remain at high utilization where the energy cost of transactions is minimized. The remaining servers can be suspended (S3) or powered off (S5). Figure 6-11 illustrates energy proportionality through migration and consolidation. With migration and consolidation represented by this example, energy efficiency can be improved by more than 25%.

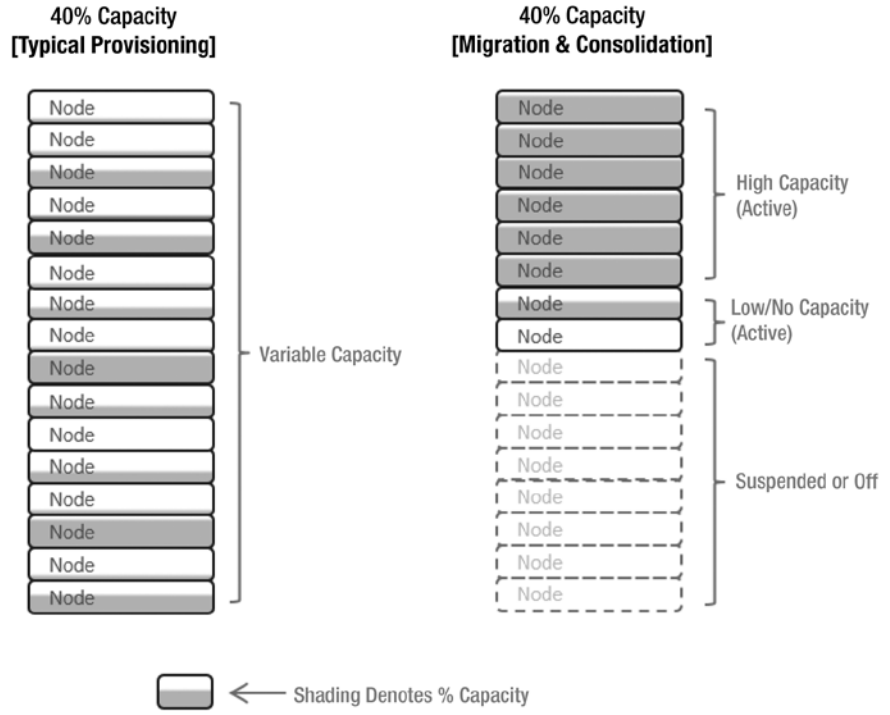

Figure 6-11. *Example of efficiency through migration and consolidation*

Inactive servers can be awakened by wake-on-LAN packets or by standard out-of-band management interfaces such as the Intelligent Platform Management Interface (IPMI). It is also common to keep some number of servers idle, but not suspended. When new VMs are created, these instances can be provisioned on unutilized but active servers, hiding the latency of initializing suspended servers. These practices allow for a more dynamic resource pool where energy efficiency is maximized and fluctuations in throughput are accommodated without waiting for servers to enter and exit S-states.

Migration is gaining adoption to improve utilization and energy efficiency, but the practice is not widespread today. There are a number of reasons why adoption is inhibited. First, VM migration is very expensive from a latency perspective. It can take several minutes to migrate a VM depending on the workload, application, and active memory footprint. During the migration, the average response time of transactional workloads increases sharply. For computational workloads, the maximum throughput decreases. Applications utilizing local storage in either case present additional challenges. Many datacenters lack the capability to accurately predict current and future demand. Transitions in and out of S-states exhibit high latency. Exiting S3 can take several seconds to resume where there is significant execution context that needs to be restored. Hardware and software innovations will continue to accelerate VM migration time and decrease S-state transition time making this a more viable energy efficiency strategy moving forward.

Comparison of Operating Environments

All OSes use the same mechanisms for power management discovery and control, however the strategy and use of these interfaces is very different. Each OS makes a unique set of decisions on how to measure activity, how frequently to change power states, and what is acceptable performance impact. Each OS determines whether these decisions are self-contained or influenced by outside services, management, and orchestration software. Each OS determines the amount of customization and tuning it will allow and how different combinations of OS decisions map to higher level global power policies.

The following sections provide details on unique characteristics of each of the most broadly deployed server operating systems. The section outlines unique traits and behavior of OSPM default settings including a look at the balance between power and performance. Major feature and policy changes of each OS are outlined to compare specific OS versions with different power management features and capabilities described in this chapter.

Microsoft Windows Server (including Hyper-V)

Windows Server OSPM is optimized for best energy efficiency by default. Applications running in this environment consume less energy, minimizing cost. Significant increases in response times are realized as a result of the focus on lower power. As is the case across all operating systems, OSPM policies do not typically impact maximum performance. Power management features that impact performance are not used when there is sustained high CPU utilization.

The Windows Server P-state policy is capacity driven, increasing frequency when utilization passes a predefined threshold. The threshold to increase frequency is high, ensuring that use of higher frequency only happens when the server is no longer able to accommodate demand at the current frequency. The policy is more aggressive in decreasing frequency when utilization decreases than it is in increasing frequency when utilization increases, leading to lower power. The Windows Server P-state policy does an excellent job of utilizing the full range of ACPI exposed P-states. Utilization is observed over tens of milliseconds and the OSPM P-state policy has the ability to maintain past history of utilization.

The Windows Server C-state policy is simple, examining utilization over a window of tens of milliseconds. The C-state policy makes a single target state decision based on the last observation window that is used throughout the current observation window. The simplicity of the policy has the advantage of being non-intrusive, adding no latency to the C-state entry path in software. With hardware demotion mechanisms filtering out non-optimal C-state decisions, the solution provides outstanding energy efficiency.

Windows Server supports user-configurable global power policies including *power saver*, *balanced*, and *high performance*. Each of the power policies represents a combination of individual OSPM parameters that control C-state and P-state policy decisions. The behavior of each of these parameters and features is highly configurable. For example, it is easy to set a P-state floor or ceiling, modify thresholds for frequency increase or decrease, or change the duration of the observation window used to measure utilization. An example of this is in high-performance mode where P-states below the advertised frequency of the CPU are not used.

It is typically much easier to tune OSPM policies to decrease response time than it is to tune policies to decrease power, making the Windows Server default behavior a good starting point for administrators looking to fine-tune OSPM to meet specific performance requirements. The power policies also change IA32_ENERGY_PERF_BIAS, adjusting hardware power management features to meet the desired balance between power and performance specified by the OS. Several of the key Windows Server tuning mechanisms are described in detail in Chapter 7.

Windows Server also includes advanced power features that can provide additional benefits in some special cases. These advanced features can provide power saving with a subset of specific workloads running on specific systems. Where feature benefits don't apply to most workloads and systems, the advanced features are typically disabled by default. The *core parking* feature in Windows Server consolidates execution to improve energy efficiency. It dynamically adjusts the number of logical processors used for running software threads, allowing some logical processors to enter deep, uninterrupted idle states. The Windows Server utility distribution feature can be coupled with core parking. Utility distribution monitors activity that cannot easily be relocated from one logical processor to another, such as software threads or interrupts affinitized to a specific logical processor. Windows Server uses this information to improve core parking decisions and to improve prediction of future demand. The *memory cooling* feature consolidates memory references to a limited set of memory power domains, saving power for systems with large memory capacity where significant portions of memory are not frequently utilized.

OSPM is largely the same between Windows Server and Hyper-V, but the specific parameters and tuning values that define power saver, balanced, and high-performance power policies vary slightly between the two. Hyper-V includes some minor changes that trade off some of the power savings for improved response times.

Table 6-10 identifies major power management features and improvements added to Windows Server over time.

Table 6-10. *Historical Changes in Microsoft Server Operating Environments*

Version	Released	Changes
Windows Server 2003	2003	Added support for C-states.
		Added support for P-states.
		Added support for T-states.
		Added global power policy (user selectable).
		Default policy is high-performance (results in P-states not used by default).
		No independent logical processor control for P-states and C-states, only a single processor supported.

(*continued*)

Table 6-10. (*continued*)

Version	Released	Changes
Windows Server 2003 R2	2005	No significant changes to Windows Server 2003.
Windows Server 2008 (and Hyper-V role)	2008	Default processor performance policy is balanced (results in P-states used by default). Added power policy for ASPM.
Windows Server 2008 R2 (and Hyper-V role)	2009	New processor power management policies with significant energy efficiency improvements. Added timer tick coalescing. Added intelligent timer tick distribution (tick skipping). Added core parking. Added power metering and budgeting. Added remote power management and group policy. Hyper-V now built in. Hyper-V live migration.
Windows Server 2012	2012	Added logical processor idling. Added memory cooling. Hyper-V supports more VMs and more processors per VM (4 to 64).
Windows Server 2012 R2	2013	No significant changes to Windows Server 2012.

Linux Distributions (including KVM)

Linux OSPM is optimized for low latency. Applications running in this environment have improved performance, minimizing transaction response times. A significant increase in power is realized as a result of the focus on lower latency. Similar to other operating systems, the OSPM policies do not impact maximum throughput. Power management features that may impact performance are not used under sustained high CPU utilization.

Linux supports P-states through the CPUfreq infrastructure which provides interfaces for low-level control drivers and high-level control policies, called *governors*. Governors can be dynamically changed, but the default for most server distributions is *ondemand*. Ondemand is capacity driven, and selects the highest available frequency when utilization is above a predefined threshold. When utilization falls below the threshold, the next lowest frequency is used. Under variable loads, frequency is increased more aggressively than it is decreased, leading to low latency. When a server is partially utilized, ondemand tends to use a limited range of ACPI-exposed P-states, with the majority of requests being for P*n* or P0. This is due to the P0 being the only target state

used whenever utilization exceeds the threshold. Ondemand observes utilization of tens of milliseconds and does not consider past history.

For environments without response time requirements that can tolerate additional latency, the *conservative* governor is an alternative to ondemand. This governor provides a more balanced use of the full range of P-states. In comparison to ondemand, the conservative governor results in higher response times, but with lower power. The *performance* governor can be used to permanently run at the highest frequency, and the *powersave* governor can be used to permanently run at the lowest frequency.

Intel also provides its own native P-state driver called `intel_pstate`. This driver is optimized for low response times and minimizes latency and the software overhead of P-state selection as the governor and scaling driver are combined. The driver is enhanced to understand the specific capabilities of each processor, which allows for improved use of Turbo. The `intel_pstate` driver can be described as a native driver, meaning it uses CPU interfaces to determine a richer set of information about power management capabilities instead of using ACPI. Native drivers are resilient to any issues the BIOS may introduce with incorrect ACPI objects.

Similar to P-states, Linux supports C-states through the CPUidle infrastructure, which provides the same separation between low-level control drivers and high-level control policies. The default governor for most server distributions is *menu*. OSPM policy uses a number of different metrics to make an optimal C-state decision. These include previous C-state residencies, expected idle duration, and the exit latency of target C-states. A target C-state is selected for every idle entry on every logical processor rather than determining a single target C-state for all logical processors over some time window. The advantage of this approach is that poor decisions are less frequent; the disadvantage of this approach is the cost of additional software overhead. The advantages typically outweigh the disadvantages, providing superior energy efficiency. Intel also provides its own low-level `intel_idle` driver that is enhanced to understand specific capabilities of each processor. `intel_idle` is able to expose more hardware C-states than an ACPI BIOS can expose to `acpi_idle`. For example, on some servers, `intel_idle` is able to export a C1 state with lower latency than C1E. Unlike `intel_pstate`, it does not replace existing higher-level policy.

Linux does not include global power policies that automatically change governors, their tuning, and platform-level controls such as IA32_ENERGY_PERF_BIAS, with a single setting. Rather, Linux relies on individually selecting and tuning C-state governors and P-state governors along with utilities such as *cpupower* to manage IA32_ENERGY_PERF_BIAS settings and options for execution consolidation. Linux has extensive capabilities to fine-tune individual power management parameters such as setting a minimum frequency or restricting use of a specific C-state. These are discussed in greater detail in Chapter 7.

Linux has several advanced power features that can provide additional benefits in certain environments. These features may work well with a specific workload and specific microarchitecture, but not well in others. As a result, some of these features are disabled by default. The scheduler supports core parking through cgroups and the CPU hotplug infrastructure. These features can be used to consolidate execution to a specific subset of logical processors. However, it must be coupled with additional management software

to enable dynamic execution consolidation. Energy efficiency benefits of execution consolidation are more common when coupled with a low-latency P-state policy, such as ondemand. Energy efficiency benefits are less common when this technique is coupled with a balanced or low-power P-state policy.

■ **Note** The kernel-based virtual machine (KVM) inherits key power management functionality from Linux, so there are very few differences in power management capabilities or policies between a native and virtualized environment using KVM. Xen does not inherit key power management functionality in the same way. Power management features added to the Linux kernel have to be re-implemented or ported to Xen, so many of the features and explanations in this section do not directly apply to Xen in the same manner.

Table 6-11 identifies major power management features and improvements added to the Linux kernel over time. In some cases, key power management features have been back-ported to add the support to existing Linux distributions. Rather than cover every distribution, along with the kernel major and minor version numbers, the Linux reference introduces the kernel version in which a capability was first introduced.

Table 6-11. *Historical Changes in Linux Server Operating Environments*

Version	Released	Changes
Kernel 2.4.22	2003	ACPI built-in P-states disabled by default
Kernel 2.6.5	2004	Added CPUfreq subsystem Added ondemand and other governors P-states enabled by default (using ondemand governor)
Kernel 2.6.18	2006	Added support for deep C-states (I/O port)
Kernel 2.6.19	2006	Added MWAIT support for deep C-states Added APERF/MPERF feedback used for P-states
Kernel 2.6.23	2007	ACPI OSI (Linux) disabled by default
Kernel 2.6.24	2008	Added tickless idle (CONFIG_NO_HZ) Added CPUidle subsystem
Kernel 2.6.30	2009	Added USB HID autosuspend
Kernel 2.6.32	2009	Added `acpi_pad` (processor aggregator device) for power limiting

(*continued*)

Table 6-11. (*continued*)

Version	Released	Changes
Kernel 2.6.35	2010	Added `intel_idle` (Intel CPUidle C-state driver)
Kernel 2.6.36	2010	Added support for deep C-states in CPU offline Added USB mass storage autosuspend
Kernel 3.1	2011	Added kernel support for IA32_ENERGY_PERF_BIAS (kernel sets this to 6 if found it is 0 at boot-time)
Kernel 3.5	2012	Removed `sched_mc_power_savings` scheduler tunable (early dynamic execution consolidation implementation)
Kernel 3.9	2013	Added `intel_pstate` (native Intel P-state driver) Added idle injection driver
Kernel 3.10	2013	Added full tickless (CONFIG_NO_HZ_FULL) allowing Linux to be built with no clock ticks, either when idle or busy
Kernel 3.11	2013	Added native RAPL driver for in-band power limiting

VMWare ESX and ESXi

In contrast to the approach taken by other operating systems, a key focus for ESXi-based environments is managing power at a cluster level, between a larger numbers of servers. Distributed Power Management (DPM) consolidates active VMs to a subset of servers, running that subset of active servers at higher utilization, while placing unutilized servers in a standby or off mode. DPM can be configured to run VMs on the smallest subset of servers possible to achieve energy proportionality.

OSPM policies for C-states and P-states are covered by Host Power Management (HPM). With multiple VMs competing for system resources, ESX Host Power Management is optimized for low latency. Applications running in this environment have improved performance, minimizing transaction response times. Applications that run in this environment have lower response times, but that comes at the cost of higher power. Similar to other operating systems, power management features that may impact performance are not used under sustained high CPU utilization.

ESXi supports user-configurable power policies including *high performance*, *balanced*, *low power*, and *custom*. These policies include both a combination of power management parameters and associated tuning values as well as static enabling and disabling of features. For example, in high-performance mode, P-states and deep C-states are completely disabled. In balanced and low-power modes, both P-states and deep S-states are enabled. Low-power mode enables all power management features and includes more aggressive use of the lower power and higher latency states. Custom policy allows administrators to specify power management parameter values such as limiting C-states based on their exit latency or changing the observation window used to measure utilization.

Table 6-12 identifies major power management features and improvements added to ESXi over time.

Table 6-12. *Historical Changes in VMWare ESX/ESXi Operating Environments*

Version	Released	Changes
VMware ESX 3.5	2007	Added Distributed Power Management (dynamic migration of VMs, shutdown and restart of servers to manage power).
VMware ESX 4.0	2009	Added support for P-states (disabled by default).
VMware ESX 4.1	2010	Added global power policy (user selectable). Default policy is high-performance (results in P-states not used by default).
VMware ESXi 5.0	2011	Added Host Power Management. Default policy is balanced (enables P-states by default, but no C-states).
VMware ESXi 5.1	2012	No significant changes.
VMware ESXi 5.5	2013	Added C-state support to balanced policy.

Summary

Operating systems play a key role in selecting both idle and active power states for the server. This is a difficult balancing act because the OS decisions heavily impact both performance and power. In addition to power state selection, OS process scheduling, I/O, interrupt handling, and memory management decisions also have a significant impact on power. Virtualized environments include many of the same capabilities as native environments. They also enable new usage models, such as migration and consolidation, which can provide substantial improvements in energy efficiency.

The list of historical changes across operating systems at the end of the chapter serves as a reference to highlight both the power management limitations of legacy operating systems as well as the latest power management enhancements in modern operating systems.