CHAPTER 5

BIOS and Management Firmware

Thousands of times a second, CPUs, memory, system interconnects, and other components transition between a number of different active and idle power states to minimize the use of energy. Energy-efficient use of these power states isn't possible without careful coordination between hardware and software—including BIOS firmware, management firmware, operating systems, and applications. If any one of these firmware or software components fails to fulfill its role in this coordination, it can cause a wide variety of problems—from increases in power to decreases in performance. Figure 5-1 illustrates the hierarchy of software components used in enabling and controlling the use of power management features.

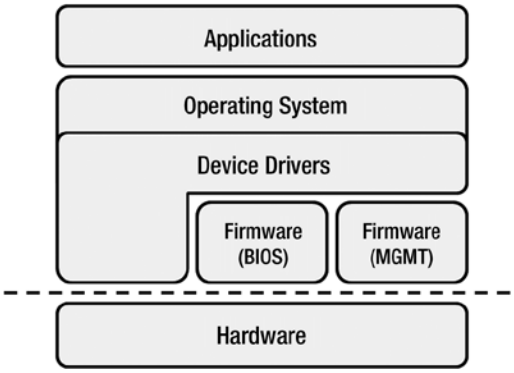

Figure 5-1. *Hierarchy of software components used in power management*

BIOS firmware is responsible for turning on and configuring power management features. BIOS must also expose power management features to the operating system (OS) to allow for software control. This advertisement includes a list of the supported power states, each state's power and performance characteristics, and a description of control interfaces. In some cases, OS device drivers can discover and configure power management features directly. The OS is responsible for monitoring the system at runtime and using the BIOS-provided interfaces to control power management features based on past, current, and projected future activity. This activity is ultimately generated by applications that use system resources to perform computations and manipulate data.

This chapter begins with a description of BIOS firmware and its role in activating and configuring features. It continues with an overview of BIOS firmware's role in updating microcode and creating Advanced Configuration and Power Interface (ACPI)) objects that describe power management capabilities to software. It includes an overview of management firmware including hardware protection, power capping, and system monitoring functions. The chapter concludes with a description of the Intelligent Platform Management Interface (IPMI) and how it is used to configure and control firmware capabilities used for power management.

BIOS Firmware

When a server is powered on, power management features such as C-states, P-states, interconnect power states, and memory power states are not configured or enabled. This is unlike many of the other system functions that do not require explicit firmware or software enabling. Enabling and configuring power management features is the responsibility of BIOS firmware.

Many power management features exist in different processor units or different system components. These features have different clock and power domains, different initialization sequences, and different enabling requirements. As a result, configuration and enabling takes place across multiple stages where BIOS firmware coordinates with the CPU's power control unit (PCU) and other units to initialize individual features. If this initialization fails, power and thermal management will not function properly.

During the various power management initialization stages, there are two common methods for communication between firmware and hardware. The first is reading and writing small data arrays in hardware called *registers*. Register size is typically measured in bits, for example, a 32-bit register or a 64-bit register. Registers provide a convenient mechanism for software and hardware communication and are used extensively in the control and configuration of power management features. The three most common types of registers used for power management are model specific registers (MSRs), control and status registers (CSRs), and memory-mapped input/output (MMIO). These registers can only be read or written in Ring 0, meaning they are only accessible to kernel mode software with the highest privilege level. Where there is a need to use a register to frequently access status information, make repeated changes to configuration settings, or regularly change power states of the processor, MSRs are used. These registers can be accessed with low software overhead using dedicated RDMSR and WRMSR instructions. Where there is power management control or status information that needs to be accessed infrequently or only during boot time, CSRs and MMIO are used. CSRs are

registers mapped to memory locations in legacy PCI configuration space, where several layers of device drivers may be required to access these after control has been passed to the operating system.

A second communication method between software and hardware is the x86 CPUID instruction. The CPUID instruction provides a fast mechanism for software to query the processor to determine feature support and configuration information, and it is accessible in Ring 3 or by user mode software (for example, the CPU type, the topology of cores and threads, and various feature support flags). The CPUID instruction provides a low-latency and error-resilient way to determine if a processor supports power management features such as P-states or C-states, and what happens to various clock sources when the processor is idle.

Not all power management features are under direct operating system control. Chapter 2 details how Turbo allows the processor to operate above the CPU base frequency, how a multi-socket system enters a coordinated package C-state, and how power and thermal events can trigger processor throttling. These types of features are controlled by CPU microcode and PCU firmware. Activity generated by applications or OS power management policies may influence the use of these features, but the core functionality is controlled elsewhere. In addition to the PCU, other microcontrollers in the system, such as a baseboard management controller (BMC) or a Management Engine (ME) in the chipset, may also participate in the control of power management features.

Microcode Update

There may be cases where it is necessary to change the behavior of hardware power management features—to fix issues, to add new functionality, or to optimize feature use. The majority of these updates can be done through BIOS firmware updates. The greatest benefit of firmware updates is the ability to improve system behavior without having to replace any components. However, the downside is the need to restart the system in order to do so. In datacenters with tens of thousands of servers, a simple firmware update becomes an event with significant cost and complexity.

Where there is a need to enhance or correct CPU-specific behavior, this can be performed by the operating system. Operating systems include updated CPU microcode and have the ability to update CPU microcode without needing to install and validate a new BIOS firmware image or reboot the server. CPU microcode updates are done by loading microcode into memory and writing the address of the microcode to the IA32_UCODE_WRITE MSR.

Throughout this book, several of the MSRs outlined start with the IA32_ prefix. This prefix indicates that the MSR is architectural, meaning it is supported in future CPUs using a consistent address and data field definition. Data fields that are reserved or undefined can be used to add new functionality over time. Architectural MSRs also do not change across different product segments. For example, the definition of IA32_PERF_CTRL is identical across phones, tablets, notebooks, and servers. This is critical for maintaining forward and backward compatibility with the various software components utilized in power management. MSRs are documented in detail in the Intel Architecture Software Developer Manuals.

Advanced Configuration and Power Interface

As discussed in Chapter 2, C-states and P-states, or processor idle power and processor performance states, are controlled by the operating system. After these features are configured and enabled by BIOS firmware, BIOS firmware is responsible for advertising these power states and control interface information to the operating system. To accommodate compatibility and flexibility between different hardware and software implementations, an industry standard interface called Advanced Configuration and Power Interface (ACPI) is used.[1] ACPI provides an interface for conveying power state information in abstract terms so operating systems and hardware power management features can advance independently without causing any loss of functionality. For example, a new hardware C-state released in 2015 can be used efficiently by an operating system released in 2005 as long as that C-state is described using ACPI. This abstraction is necessary because there can be significant differences in power state behavior between different platforms, architectures, and CPUs, even when those power states share the same name. Similarly, OS power management policies may choose not to use a power state in one product and version, but may choose to in another. The scope of ACPI goes beyond describing power states and control interfaces. Rather than a comprehensive review of ACPI capabilities, this section discusses only those key interfaces most relevant to understanding the hardware and software interaction in power management.

ACPI provides an abstraction for several different types of states including S-states, C-states, P-states, T-states, and D-states. ACPI states are identified by a letter indicating the state type, followed by a number indicating the depth of the state. For example, S0 is system state 0 and P5 is performance state 5.

Lower numbers indicate a state with more activity and higher power—the number 0 always indicates the state with the most activity, highest performance, and highest power. As a result, lower numbered and higher power states are called *shallow power states* whereas higher numbered and lower power states are called *deep power states*. The description of states as shallow or deep is done to convey state transition costs, such as latency or the transitional energy needed to enter or exit a state. During these transitions, the system is stalled or may be taking actions that will affect performance when the processor resumes execution, such as flushing caches and translation lookaside buffers (TLBs).

A resource can only be in one state type at any given time. For example, a system can only be in one S-state at a time, a core can only be in one P-state at a time, and a device can only be in one D-state at a time. Figure 5-2 illustrates the relationships and dependencies between S-states, C-states, and P-states and how a system transitions between them.

[1] Advanced Configuration and Power Interface Specification, Revision 5.0a, November 13, 2013.

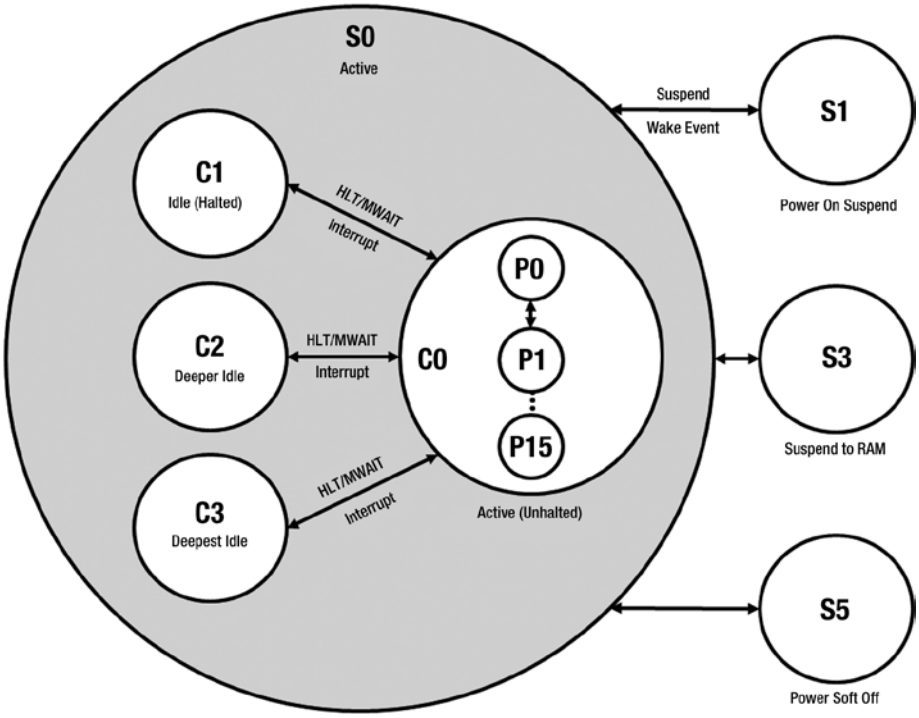

Figure 5-2. *Summary of ACPI power states and transitions*

S-states

S-states refer to system level sleep states and include S0, S1, S3, S4, and S5. The CPU only executes instructions in the S0 state. The use of other S-states is somewhat uncommon for servers, because most servers are usually in an active state (S0), where they are active or ready to execute or they are powered off (S5). A server in an S5 or soft off state is one that is powered off, but still plugged in. Even though an ACPI S0 state describes an active state, it is possible for processors, devices, or other resources in the system to be idle. System level states can be in a shallower state than processor or other device states, but they can never be in a deeper state.

ACPI S3, commonly referred to as suspend, is a sleep state where OS context is saved to system memory; memory remains powered, but most of the other system components are powered down. In S4, all devices have been powered off, but current OS context has been retained on a storage device. S3 and S4 support varies with many server products not supporting these.

■ **Note** Use of S3 and S4 is uncommon in servers. These states do not maintain an active network connection, and execution context is no longer in CPU caches. It can take a significant amount of time to resume from these states, making them difficult to use in dynamic environments with variable load.

C-states

The ACPI specification defines three types of idle power or C-states: C1, C2, and C3. A C0 state describes an active processor that is executing instructions. An ACPI C1 state is mandatory. It describes the lowest latency idle power state and is reserved for processor states that have an insignificant amount of transition latency or performance impact. An ACPI C2 state is a deeper state than C1, with lower power and higher latency. It's allowed to have measurable latency impact but does not require any additional software handling above what an ACPI C1 state requires. An ACPI C3 state is the lowest power and highest latency state and has extra software overhead associated with C-state entry and exit. ACPI C3-type states have software visible effects. Use of these states may require the OS to check on chipset activity before entering the state or may require the OS to identify and use alternative time sources due to a processor timestamp counter or local APIC timer stopping after entering the C-state.

Due to the increased software complexity of ACPI C3-type states, most modern servers do not implement C-states that map to anything deeper than an ACPI C2-type state. Over time, hardware C-states have been optimized to eliminate or reduce software visible effects. This ranges from architecting timers so they continue to run when the processor is in deep C-states and eliminating dependencies on activity level outside the CPU to aggressively reducing deep C-state exit latencies.

■ **Note** A common point of confusion is the difference between hardware C-states and APCI C-state types. Each of the hardware C-states described in Chapter 2 is mapped to an ACPI C-state type when they are advertised by BIOS firmware. For example, hardware C1 states map to an ACPI C1 type whereas hardware C3 and hardware C6 states map to an ACPI C2-type state.

P-states

P-states refer to processor performance states and include ACPI P0, and P1 to P*n*, where the number of states between 0 and *n* varies based on the number of unique voltage and frequency operating points supported by the processor. P*n* is also referred to as the deepest P-state. Unlike S-states or C-states, which represent idle states, P-states represent active states, and as a result, ACPI P-states are only utilized when the processor is in C0, actively executing instructions.

The P0 state is the highest performance and highest power state, and every state from P1 down to P*n* results in a decrease in power and a decrease in performance in comparison to lower-numbered states. ACPI limits the number of P-states to no more than 16. In cases where a processor has more than 16 hardware P-states, BIOS firmware must decide which of these are exposed to the OS. BIOS firmware typically exposes an ACPI P-state for every base clock step between the processor's minimum frequency (P*n*) and the CPU base frequency. Turbo mode, discussed in Chapter 2, is always mapped to ACPI P0.

D-states

The ACPI specification also defines device power states, or D-states. ACPI D-states aren't utilized as frequently on servers as they are in clients. Many servers are unable to use D-states since the latency to resume from these states is too significant for use in active servers. Another reason ACPI D-states aren't always exposed on servers is because additional standard interfaces for device state discovery and control exist, such as Power Management Control and Status register (PMCSR) , defined by the PCI and PCI express specification. Some device drivers manage native device-specific power states via private device-specific controls. Many devices have the capability to monitor their own device activity and manage power without any software control. Even where there are no software exposed D-states, devices or CPUs may be autonomously transitioning between various power states at runtime to manage power.

Although there are various control methods and specifications that describe device power management outside of ACPI, they all share the same terminology. A D0 state is active, D1 and D2 are low-power states where device context is saved, and D3 is the lowest power, highest latency state where no device context is saved.

ACPI Interfaces

An operating system needs a much greater level of detail about power states and their behavior in order to utilize them efficiently. When selecting between different power states, the OS needs to know each state's power consumption, the transition time to enter and exit a state, what level of execution context is lost upon entering a state, and specific mechanisms for initiating entry. ACPI standardizes tables to describe this detailed information to the OS.

BIOS firmware is responsible for constructing these tables and loading them into memory where the operating system reads them and enumerates capabilities. ACPI tables that include core power state information are the DSDT (Differentiated System Description Table) and the SSDT (Secondary System Description Table). These tables consist of several objects that provide needed power state and control information to the OS. The following list describes the primary set of ACPI objects used by the OS:

- **_OSC and _PDC (Operating System Capabilities and Processor Driver Capabilities):** These methods are used to communicate capabilities of the OS to BIOS firmware. This includes describing OS capabilities in terms of coordinating control across multiple logical and physical processors, and its ability to control P-states and C-states. The capabilities of the OS will determine what power management features BIOS firmware will expose.

- **_PSS (Performance Supported States):** This object lists the P-states available to the OS. For each state, the object includes a performance level (typically core frequency in MHz), the maximum power consumption, and transition latency. In addition, the object lists a control register value that the OS uses to identify a P-state when requesting a power state change and a status value that the OS uses to identify a P-state when checking on processor status. A separate ACPI object defines the processor status register that the OS uses to determine the current P-state and to check the status of existing P-state control requests.

  The ACPI specification describes P-states in terms of guaranteed frequency. Turbo mode, or the ability for processor cores to run above the CPU base frequency, is not guaranteed. Turbo is opportunistic with the frequency dependent on thermal or power headroom. Since Turbo frequency is not guaranteed, it is improper to expose the maximum Turbo frequency via ACPI. As a result, the ACPI _PSS exposes Turbo (P0) at 1 MHz higher frequency than P1. When the OS requests Turbo, hardware will maximize frequency, potentially running well above what is advertised to the OS. If frequency determinism is a hard requirement for users, software interfaces are provided so Turbo can be disabled.

■ **Note** There are many cases where exposing Turbo as a single P-state is not energy efficient. Not allowing the operating system to choose intermediate P-states between P1 and the maximum Turbo frequency can result in selection of a P-state that is higher performance and power than required.

- **_PSD (P-state Domain):** The ACPI _PSD object describes CPU control dependency, defining whether logical processors in a CPU have their own P-states or whether subsets of logical processors share a common P-state. Some server CPUs have a single P-state domain, meaning all logical processors in that CPU share a single frequency. Other server CPUs have a different P-state domain for each core, meaning that all cores in the CPU can run at their own independent frequency. The _PSD object is also responsible for describing the P-state coordination type, which is discussed in greater detail later in this chapter.

■ **Note** Errors in ACPI objects can cause significant power issues. Figure 5-3 shows the impact of an online transaction processing workload running on a system with an incorrect _PSD. These issues have resulted in substantial (up to 40 W higher) power increases throughout a range of different operating conditions.

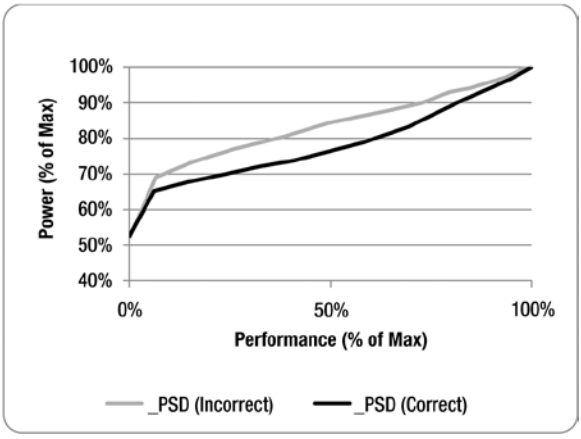

Figure 5-3. *Power impact from ACPI _PSD object with an incorrect mapping*

- **_PCT (Performance Control):** This object describes the processor registers or firmware locations that allow the OS to change P-states and to check the status of P-state requests. To change P-states, the OS uses the _PCT-specified control register. P-state transitions are initiated by the OS writing a P-state's control value (specified in the _PSS) to the control register. In order to check the status of P-state requests, the OS uses the _PCT-specified status register. The status register also specifies the current P-state in terms of the _PSS-specified P-state control value.

  Most modern processors specify these interfaces in terms of what ACPI calls Functional Fixed Hardware (FFH) , or a processor MSR. This allows the performance control interface to be implemented directly in hardware providing a low latency and error resilient interface. Where a native processor interface is not available or desirable, an original equipment manufacturer (OEM) can implement platform-specific code to handle performance control.

- **_CST (C-states):** This object lists the C-states available to the OS. Details provided for each C-state include the register used to place a processor into a C-state, the ACPI C-state type, worst-case latency, and typical power consumption. Power consumption numbers in the _CST are not used by operating systems because they are assumed to be estimates only. The latency field is used by several OS control policies to limit use of some C-states based on system activity levels or where there is a specific device that can't tolerate latency above some threshold.

  Similar to ACPI P-state objects, an ACPI _CSD object exists to describe processor control dependences for C-states. Understanding cross logical-processor C-state dependencies is useful for understanding C-state impact when the OS needs to consolidate execution to some subset of logical processors in the CPU.

After an operating system initializes, it evaluates these ACPI methods and has all the information it needs to request hardware transitions between various idle and active states and to check the status of those requests. The OS control policy uses ACPI-advertised information outlining the expected power and performance impact of the various states to make state transition decisions, and it uses ACPI-advertised control mechanisms to execute those decisions.

■ **Note** There is a long history of issues with the resiliency and robustness of the ACPI interface for OS power management. Some modern operating systems are starting to use native processor interfaces such as CPUID to discover the CPU type, power management features, and control interfaces directly from hardware. This use of native processor interfaces limits how flexible the power management solution is, but it eliminates errors in ACPI objects from causing functional or performance issues.

Setup Utility

The setup utility, also implemented in BIOS firmware, is a powerful tool for fine-tuning power management and optimizing a platform for specific workloads. The majority of power management features can be enabled, disabled, or have their default behavior changed through simple setup options. Chapter 8 is a comprehensive optimization reference that discusses options commonly found in the setup utility and different ways these options can be configured to decrease power, increase performance, or, in an ideal scenario, both.

Management Firmware

Microcontrollers in cars can monitor fuel level and consumption rate. They can indicate if tire pressure is low or if a turn signal is burned out, and they can keep a record of diagnostic events that can be retrieved by a technician during a service appointment. These capabilities are provided independent of who is driving the car, or if the car is speeding or is stopped.

Management firmware running on server microcontrollers plays a very similar role. Management firmware provides power, monitoring, event logging, inventory, and remote management capabilities independent of the OS or state of the processors. This is particularly useful in the datacenter where there is large number of servers, where systems are going up and down for maintenance, and where servers are running different operating systems. Two key firmware components that are critical for power management are the baseboard management controller (BMC) firmware and the Management Engine (ME) firmware, called Node Manager.

Node Manager Capabilities

Node Manager firmware provides key capabilities for managing and optimizing both power and cooling resources in the datacenter. It exposes a standardized set of hardware protection, monitoring, and power capping features to the BMC and to external management software. Node Manager acts as a satellite controller and offloads power management responsibilities from the BMC, with some of the capabilities always running and others activated by a profile.

Hardware Protection

Node Manager firmware implements a set of hardware protection mechanisms to protect the platform during adverse or unexpected conditions. There are proactive protection mechanisms such as dynamically limiting platform power to the capabilities of the PSU. There are also reactive protection mechanisms such as closed loop system protection (CLST) and Smart Ride Through (SmaRT) that protect the platform during PSU over-temperature, under-voltage, and over-current events. These capabilities are hardware assisted, with sensor devices using the SMBUS protocol to notify Node Manager about critical events. Protection mechanisms respond immediately in the case of an adverse condition, with required actions, such as processor and memory throttling, occurring in under a millisecond.

Monitoring

Another key capability of Node Manager firmware is comprehensive platform monitoring. In addition to the monitoring capabilities provided by processors and memory, modern servers implement several onboard sensors in intelligent power supply units (PSUs), voltage regulators (VRs), hot swap controllers (HSCs), and in devices accessible by the BMC. These sensors enable fine-grained power monitoring since they are capable of reporting voltage, current, power, and energy consumption for individual components. Combining together all the board, processor, and memory sensors creates a sensor grid that Node Manager firmware relies on for power management.

These monitoring capabilities have uses beyond enabling Node Manager's protection and power capping features. External management software uses these monitoring capabilities in a variety of ways. For example, events that monitor inlet temperature, outlet temperature, and volumetric airflow are used by facility control software.

■ **Note** In order to expose more information about the platform, Node Manager adds several synthetic sensors such as outlet temperature and volumetric airflow. These sensors are derived from other sensors in the platform and calculated based on a mathematical model.

Events that monitor compute utilization and memory utilization are used by orchestration software to aid in workload placement and migration decisions. Events that monitor power consumption are used to characterize and optimize production workloads. Various types of sensors and usages are described in greater detail in Chapter 7.

Power Capping

Node Manager firmware allows users to set and enforce a power cap ensuring that power will not exceed a defined threshold. External management software uses this capability in several different scenarios to provide power, performance, and cost benefits.

Most servers operate well below the theoretical maximum platform power, even when workloads are running at peak throughput. This limits the number of servers that can be safely added to a rack with fixed power capacity. Rather than allowing a server to operate up to the theoretical maximum platform power, users can enforce a power cap that corresponds to more representative peak conditions. This power cap can be determined using insight gained from datacenter monitoring, it can be established by characterizing production workloads under peak conditions, or it can be established based on some percentage of the theoretical maximum platform power. Using a more representative power cap, rack density can be improved.

■ **Note** Node Manager provides a feature that automatically characterizes platform minimum and maximum power during BIOS POST using specialty workloads. The results of this characterization can be used to identify an appropriate power cap when it is not possible to do so using production workloads.

Several additional applications of power capping exist—for example, power capping to survive a power or cooling failure in the datacenter. An aggressive power cap can be enforced during these conditions, decreasing server power and cooling requirements. This keeps applications running, delaying or avoiding automatic shutdown. Power capping can also be applied strategically to maximize resources where energy has a variable cost. Figure 5-4 shows external interfaces and components used by Node Manager firmware to enable hardware protection, monitoring, and power capping.

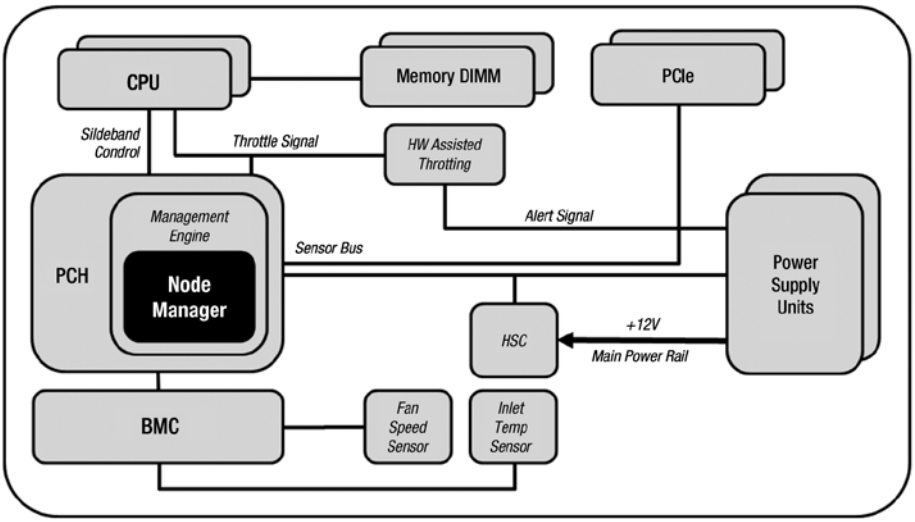

Figure 5-4. *External interfaces and components used by Node Manager firmware*

Node Manager Policies

By default, Node Manager firmware activates hardware protection and basic monitoring. Additional Node Manager firmware capabilities are supported using user-defined policies that can be created, configured, enabled, and disabled.

Policies can be always running, such as a power capping policy that replaces the theoretical maximum platform power with a cap more representative of peak conditions. Policies can also be enabled by a trigger, or some monitoring event used to activate the policy. For example, a policy can enforce a power cap only when an inlet temperature exceeds some threshold. An operator might define several different inlet temperature thresholds, with each one activating a different power cap.

Node Manager policies typically monitor or control power for a specific policy domain. A domain is simply an abstraction for related individual platform components. For example, a policy that targets the platform domain includes all components in the server. The CPU domain would report and control power for all CPUs in the system, treating them as a single entity, while the memory domain reports and controls power for all DIMMs and memory controllers in the system.

Table 5-1 lists all the different attributes of a Node Manager policy that operators can use to configure policies to match desired behavior and specific needs. These attributes enable more sophisticated event-driven management. For example, if the server is unable to meet a power cap, the policy can define a resulting action, such as sending an event to external management software or shutting down the platform.

Table 5-1. *Attributes of a Node Manager Policy*

Attribute	Supported Values	Description
Assigned policy ID	A one-byte numeric value.	Indicates a unique identifier for the policy. This is assigned during policy creation.
Policy domain	Can be any of the following: • Entire platform • CPU subsystem domain • Memory subsystem domain • Hardware protection domain • High-power I/O domain	Indicates the specific platform subsystem the policy is applied to.
Administrative state for policy	Can be either of the following: • Enabled • Disabled	Indicates the state of the policy. Even if a policy is disabled, monitoring for the policy is still enabled.
Policy trigger type	Can be any of the following: • No policy trigger. • Inlet temperature limit (in Celsius). • Missing power reading timeout (in 1/10th of a second). • Time after host reset (in 1/10th of a second). • Boot time policy. This policy will be applied only at boot.	Indicates the trigger for policy activation. If "No policy trigger" is specified, the policy is always active. For all other triggers, the policy is only active while the condition is true.
Policy trigger limit	A temperature value in Celsius or a time value in 1/10 of a second.	Indicates the specific value associated with the trigger.
Policy limit	A power cap can be specified as one of the following: • Power (in W) • Throttling level (in %)	Indicates a power cap to be enforced. Platform throttling level is used in case of missing power readings.

(*continued*)

Table 5-1. (*continued*)

Attribute	Supported Values	Description
Aggressiveness	Can be one of the following: • Automatic • Force unaggressive mode • Force aggressive mode	Indicates the types of power management mechanisms used to keep the server below a power cap. Node Manager attempts to meet a cap using the most energy efficient mechanism available. Mechanisms with greater performance impact are used only when a cap cannot be met using the energy efficient mechanisms.
Correction time limit	A time value in milliseconds.	Indicates the maximum time, in milliseconds, in which the Node Manager must take corrective actions to meet a power cap. If this time is exceeded, the "Policy exception action" specifies the next action.
Policy exception actions	Can be either or both of the following: • Send alert. • Shut down system (hard shutdown via BMC).	Indicates action taken if the policy limit cannot be met. Sending an alert will cause Node Manager to generate an asynchronous event to notify external management software that the defined limit is too low.
Policy storage option	Can be either of the following: • Persistent storage • Volatile memory storage	Indicates the storage type of a policy. By default the policies are stored persistently so Node Manager will restore the policies after each platform reset. If policies are frequently created and updated, volatile storage should be used.

(*continued*)

Table 5-1. (*continued*)

Attribute	Supported Values	Description
Statistics reporting period	A time window in seconds.	Indicates the averaging window for monitoring. This allows operators to specify up to a one hour moving average window for monitoring.
Alert thresholds	Up to three thresholds in the units specified by trigger type. For a policy without a trigger, the thresholds array contains average power in watts. For temperature-based triggers, the thresholds array contains temperature in degrees Celsius. For time-based triggers, the array contains time in 1/10 of a second.	Indicates threshold trigger values need to exceed to generate events.
Suspend periods	An array of start and stop times including recurrence patterns based on the day of the week.	Indicates when the policy will be enforced.

The policy allows operators to specify various alert thresholds. Each policy supports up to three thresholds that can be used to generate events. For example, it is common to set a threshold close to the defined power cap, so external management software can see how close the system is getting to enforcing a cap.

The policy allows operators to specify suspend periods. This defines a weekly pattern of days and times a policy should be enabled or disabled—for example, power capping servers hosting IT infrastructure during nights and weekends. Node Manager automatically synchronizes the real-time clock used for scheduling with the host OS real-time clock to keep software and systems in sync.

IPMI

BMC and Node Manager firmware capabilities are configured and controlled through the Intelligent Platform Management Interface (IPMI).[2] Support for IPMI is widespread, with the vast majority of servers supporting it. IPMI provides a standard well-defined interface between external management software and the underlying platform, enabling various monitoring, logging, inventory, and recovery functions using simple request and response messages.

[2]IPMI Specification, v2.0, Rev. 1.1.

IPMI messages or commands target the BMC. The BMC acts as a communication hub for satellite controllers in the platform that include their own monitoring and control capabilities such as the ME in the PCH. Communication between the BMC and satellite management controllers takes place over an I2C bus using the Intelligent Platform Management Bridge (IPMB) interface. The I2C bus, SMBus, PMBus, memory-mapped I/O ports, as well as private management busses are all used to connect management controllers to various sensors in the platform.

Use of IPMI eliminates the need for vendor-specific tools that are incompatible between different platforms. Since IPMI is an open standard, it also allows servers to implement management functionality independent of the OS, BIOS, or the system configuration. Monitoring functions in the BMC can be accessed by IPMI out-of-band, over the network by a connected client. These functions can also be accessed by IPMI in-band through management tools and device drivers installed on the server.

Sensor Model

Sensors in a platform, such as a CPU and memory temperature sensors are discovered by management software using IPMI commands. This discovery is aided by the IPMI sensor model. The sensor model describes all the different sensors supported, as well as each sensor's name, type, and the values they return. Some sensors may provide real-time measurements whereas others may provide only a count or indication of past events.

Sensor information is stored in IPMI sensor data records (SDRs). In addition to storing information on sensor capabilities, the SDR is used to describe the various devices connected to the ICMB and it associates each sensor with the host management controller. The SDR also provides information on event generation capabilities and describes thresholds that can be set to trigger events.

■ **Note** IPMI is extensible so server manufacturers are able to add their own custom sensors and commands. As a result, management controller monitoring capabilities can vary greatly from one server to another.

Inventory information such as FRU (field replaceable unit) devices connected to the platform are stored in the SDR. FRU data includes information for inventory management such as serial number, part number, manufacturer, and description.

System Event Log

Events generated by the BMC and by Node Manager firmware are stored in a centralized event log called the System Event Log (SEL)—for example, an indication that a fan is no longer functioning properly. Similar to SDR access, IPMI enables access to the SEL providing common functions such as reading or clearing the log.

Satellite controllers send their messages to this centralized log via the IPMB, allowing the SEL to act as a single platform event repository. Stored in flash memory, events captured in the SEL contain critical information that isn't lost if power is disconnected or there is an operating system failure.

Node Manager API

Node Manager capabilities described earlier in the chapter are accessed and controlled through IPMI commands. The Node Manager IPMI API includes numerous IPMI commands for creating policies, configuring policies, accessing monitoring capabilities, and accessing runtime attributes.

The Node Manager IPMI API describes commands operators can use to query capability and version information. External management software relies on this interface to discover capabilities as different platforms may expose a different set of policy domains and features.

The API describes commands operators can use to specify the destination of alerts. Node Manager defines a set of events that are sent directly to external management software, bypassing the BMC SEL. For these events, the IPMI Alert Immediate API is used. This gives external management software the choice between event-driven or periodic polling management. The API also includes commands that provide additional management functionality. For example, commands that enable operators to set a Turbo synchronization ratio, or a frequency limit for Turbo. This feature can be used to improve performance determinism in high performance computing (HPC) environments.

A complete list of commands included in the Node Manager IPMI API are included in the Node Manager specification at www.intel.com/content/www/us/en/power-management/intelligent-power-node-manager-specification.html.

The Node Manager IPMI API includes two types of interfaces:

- **External API:** This is designed for use by external management software. This API uses the policy domain abstraction to expose high-level monitoring and control to external management software. Exposing platform management capabilities at a domain level simplifies management and improves scalability, especially in environments with thousands of servers to manage.
- **Internal API:** This is designed for use by the BMC or other management controllers in the system. This low-level API allows the BMC to access specific sensors and control features as needed.

ACPI Power Metering Objects

In addition to IPMI, Node Manager also exposes power monitoring and power capping capabilities through ACPI power metering objects. This allows the OS to participate in monitoring and control. The key ACPI objects that enable this functionality are outlined here:

- **_PMC (Power Meter Capabilities):** This object describes the power meter capabilities including measurement unit, type, accuracy, and sampling time. It describes whether the platform is capable of monitoring platform power, enforcing a power cap, or both.
- **_PMM (Power Meter Measurement) and _PAI (Power Averaging Interval):** The _PMM object returns the latest reading from the power meter. The averaging window for the _PMM returned reading is defined by the _PAI.
- **_PTP (Power Trip Points):** This object is used to define the upper and lower trip points for the power meter.
- **_SHL (Set Hardware Limit) and _GHL (Get Hardware Limit):** These objects are used to enforce a platform power limit.

Summary

BIOS and management firmware play a critical role initializing, configuring, controlling, and advertising power management features to external software. Without these actions, systems would fail to utilize power management features, resulting in high power usage, low performance, or both. Standard interfaces such as ACPI and IPMI are used to enable firmware to communicate with external software in an OS-independent fashion.

Chapter 6 will continue the discussion on software architecture with a description of the operating system's role in power management. It will describe how the OS uses BIOS firmware exposed ACPI objects to enumerate and control various power states. It will also describe how OS power state selection, process scheduling, and memory management decisions made by the OS impact energy efficiency.