Proceedings of the Linux Symposium

Volume Two

July 19th–22nd, 2006
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

Review Committee

Jeff Garzik, *Red Hat Software*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat Software*
Ben LaHaise, *Intel Corporation*
Matt Mackall, *Selenic Consulting*
Patrick Mochel, *Intel Corporation*
C. Craig Ross, *Linux Symposium*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
David M. Fellows, *Fellows and Carr, Inc.*
Kyle McMartin

Authors retain copyright to all submitted papers, but have granted unlimited redistribution rights to all as a condition of submission.

The State of Linux Power Management 2006

Patrick Mochel
Intel Corporation
mochel@linux.intel.com

Abstract

Power Management, as it relates to Operating Systems, is the process of regulating the amount of power consumed by a computer. It is a feature that is available in every type of computer that Linux runs, though the expectations and the implementations of power management vary greatly depending the makeup of the platform and the type of software running on it.

This paper provides an analysis of power management and how it relates to the Linux kernel. It provides a complete (though not comprehensive) survey of power management features, the policy to control those features, and user constraints of the policy.

1 Power Management Introduction

Power management is the effort to minimize the amount of power used over time, as measured in watts (or fraction thereof).

Technically, a watt is defined as one joule of energy per second, but it is often used in reference to the amount of power consumed in one hour of time. For example, the statement "This computer has a 300W power supply" refers to the maximum amount of power consumed over the course of an hour. [watt]

The total power consumption of a computer that we work with in this paper is measured simply by the power consumption of each of its components. (The amount of additional power needed to compensate for inefficiencies of power supplies and dissipation is not covered.) In a computer in which the power consumption of each device remains constant over time, then total power consumption can be measured like this:

$$Cp = D1p + D2p + D3p + ...$$

However, hardware features cause the amount of power consumed by a device to fluctuate over time. So, the amount of power consumed by a device over a set amount of time (e.g. one hour) can be measured as the sum of power consumed by the device at each interval (e.g. one millisecond) during that time. So, the equation becomes a bit more involved for each device:

$$D1p = (D1p0 + D1p1 + D1p2 ... D1pn) / n$$

The rates of power consumption for a device are expressed in watts/hour, so the amount of power consumed per interval must be divided by the number of intervals sampled.

2 Device Power Management

Each device in a computer has the potential to perform a certain of work over time:

$$\text{Work} = w0 + w1 + w2 + ... + wn$$

If the power consumption of a device is constant, then it consumes the same amount of power at each interval, regardless of whether or not it is actually performing its potential maximum amount of work possible. Hardware power management features provide the ability to adjust the amount of power that a device consumes based on the amount of work that the device needs to perform. The concept is based the basic observation that devices are often underutilized—the amount of work demanded from them may be less than the supply that their potential offers. That general observation provides two more specific theories about work load and power consumption.

- A device may operate at a slower rate or experience longer periods of idleness in order to perform the work demanded of it. Please see Section 2.1.
- Part or all of a device may be disabled if there is no work for it to do. Please see section 2.5.

2.1 Operable States

Approaches to reduce power consumption of a device, yet all the device to continuously perform work are commonly implemented by CPUs, which have a near-constant demand for their resources. They accomplish this by doing one or both of the following:

- Frequency modulation.
- Low-power idle states.

2.2 Frequency modulation

The frequency is a function of the voltage coming into the device and a set of multipliers (among other things). In some cases, the multipliers can be adjusted to simply affect the speed of the device, though that doesn't offer great savings in power because the device is still drawing the same amount of current from the power source. But, by adjusting the voltage, the actual power draw changes, allowing for better power savings.

Technically speaking, the device will have a range of frequencies that it can operate at for each voltage supplied. By adjusting the voltage, what is really being adjusted is the min/max range of frequencies, given the multipliers available. Within each voltage level, the multipliers can then be adjusted to achieve the desired frequency [Pentium M].

The caveat of changing voltage levels as opposed simply changing frequency levels is that the voltage may also effect the multipliers that are used to determine the bus speed between that device and another device (e.g. between the CPU and RAM). There is a higher latency involved in changing voltages because those values must be changed and synchronized in lockstep.

By being able to operate at different frequencies at the same voltage level, and often the same frequency at different voltage levels, the power consumed can be minimized, but still allow for low-latency transitions between frequencies.

2.3 Linux support for frequency modulation

Many modern CPUs support frequency modulation, and the Linux cpufreq subsystem has developed intelligent support for predicting and

Architecture	CPUs
x86/x86-64	Generic ACPI AMD PowerNow Intel SpeedStep Transmeta Crusoe NatSemi Geode / Cyrix MediaGX VIA Longhaul
ARM	Integrator SA1100 SA1110
PowerPC	Various G3s & G3s
Sparc64	UltraSPARC IIe & III
SuperH	SH-3, SH-4

Table 1: Architectures Supported by cpufreq

adjusting CPU load. cpufreq has support for many CPUs across several architectures, as seen in Table 1. Support for a specific CPU requires a driver that understands the model-specific interface for determining and entering the supported states [cpufreq].

Other devices that experience continuous demand for resources could benefit from being able to modulate their frequency depending on the amount of demand. In particular, memory and I/O buses are both large consumers of power that could frequently experience under utilization of their resources. However, no known hardware features (let along Linux support) exist for these items. This is not surprising, as their software control would most likely involve platform-specific commands and coordination that is currently performed only by the firmware (and beyond the knowledge and interest of the kernel).

2.4 Low-power idle states

A period of idleness is a small, sometimes fixed, amount of time that a device is not being used. Some devices may be able to transparently enter a special low-power state during this period. This state effectively disables the device, but ensures that it will automatically transition out of the low-power state when it receives a request for work. This feature is most famously implemented by x86 CPUs and specified by ACPI as the Processor C states: C0, C1, C2 . . . Cn [ACPI].

Low-power CPU idle states are designed for use during the idle thread. How they implemented, and how they are used, is dependent on the platform. C1 on x86 CPUs may entered by simply executing the 'hlt' instruction. Using any low-power states beyond that requires very specific manipulation of the hardware. ACPI provides an abstract interface for doing this on supported platforms, which is implemented in the ACPI idle thread [ACPI Docs], though other platforms must do it manually [DPM WP].

Each C state has a tradeoff between the power it consumes and the latency for returning to C0. Depending on the demand for the CPU (as measured by the amount of time spent idle), a lower or higher C state may entered, since a lightly-loaded system can safely endure slightly higher latencies.

Low-power idle states are starting to be implemented by other devices that are under frequent demand, but also experiences frequent periods of idleness. In particular, some PCI Express chipsets have implemented low-power idle states for its device links (connections between the PCIe controller and downstream devices). These states are called *L States*, and may be used in two cases: when the downstream device is active but idle (called Active State Power Management); or when the downstream device is in a low-power state (called Link Power States). This feature is not yet supported by Linux. More information can be found in the References [PCIe PM].

2.5 Inoperable States

Devices that are not needed for long periods of time may be put into an inoperable power state in which power to some or all of the device is removed. These states are typically used for peripheral devices that experience long periods of idleness or are known to be unneeded for a significant period of time (in a magnitude of seconds and higher). There are two basic classes of inoperable power states: where part of the device is inoperable, and where the entire device is inoperable.

Many devices have more than one usable component on them that is not strictly necessary for basic operation of the device. For instance:

- CPU with more than one core
- A graphics device with a 2d acceleration engine and a 3d acceleration engine.
- A NIC with a TCP Offload Engine

In theory, these extra components could be turned off independent of other hardware components. In some cases, like the 3d acceleration engine, this could result in significant power savings. However, the description of these features and how they controlled is device specific, and usually kept proprietary. And, with basic device power management support still incomplete, these features are not supported under Linux.

When an entire device is not being used, it can be put into a low-power and inoperable power state. The PCI Power Management Specification defines a set of 4 states that PCI devices may support: D0 (Fully On), D1, D2, and D3 (Off).

A range was specified to allow hardware designers the ability to choose alternative implementations in which more components of the device lost power as a deeper power state was entered. This would allow a lower transition latency from the low-power state to the D0 state because an entire device reset would not be necessary. However, few hardware designers have found that tradeoff worthwhile as very devices support D1 and D2.

An exception are graphics controllers, which are the biggest users of the intermediate power states. Unfortunately, the software steps necessary to reprogram a graphics controller after it has returned to D0 from any low-power state are complicated and kept very secret.

Linux provides an interface for using low-power device states via sysfs. Each device's sysfs directory has a sub-directory named `power`, which in it has an attribute file named `state`. Reading this file returns the current low-power state of the device (0 for on, non-zero otherwise). Writing to this file places the devices into a different power state.

This sysfs interface is provided regardless of the bus that the device resides on, but is a bit unintuitive for some power management implementions: it only supports turning the device on and off, and the mechanism for doing so is by writing the ASCII characters "0" and "2" respectively, which do not map to any known bus states.

3 System Power Management

Under normal, unoptimized circumstances a computer will consume as much power as it possibly can as it performs work. However, power management can be implemented for an entire system in a manner analogous to power

management of a single device: it may enter a different operable state that conserves power but preserves the ability to perform work; or it may enter an inoperable low-power state that performs no work, but offers a relatively low-latency to return to an operable state. The unique attribute of system power management is that it requires little or no hardware support beyond what is already implemented in device power management.

3.1 Operable States

An arbitrary operable system state can be defined as a set of minimum and maximum states (operable or inoperable) for each device in that particular system. By default, the system is in an operable state in which every device has their default minimum and maximum state ranges (usually Fully On and Fully Off). However, new low-power states can be defined for the system that allow work to be performed but minimize the power usage of one or more devices.

To illustrate this, consider a user who has boarded plane with his laptop. By default, the system is Fully Operable—every device is executing at its maximum speed, though some devices may be automatically transitioned to a low-power state because they are idle. Without an explicit state to enter, the user will have to manually adjust the power state of the devices affected by the change to the plane locale: turn the wireless off, turn the sound on, and keep the CPU at a speed adequate for listening to music and editing a document. However, by defining a new operable low-power state, these items can be performed automatically by simply entering the "airplane" state. Any other system optimizations that have been defined by the distributor or the OEM may also be performed by that state transparently to the user, saving them from the burden of remembering far more details about their system than they need to.

There is little support for operable system states in Linux today. The closest thing is the Dynamic Power Management (DPM) project that defines "operating points" for a system. However, the operating points so far are more concerned with the low-level power parameters of core system components like CPUs and RAM. The origins of DPM are in the manipulation of CPU power states without a firmware abstraction layer like ACPI to mask the implementation details, so its primary goal is well understood. And, without alternatives, it is the best candidate for extension to a broader system state definition mechanism. [DPM Project]

3.2 Inoperable States

Inoperable low-power system states are also known as "suspend states," the most well-known form of power management. The concept is simple: when the system is not being used, everything is stopped and the system itself is put into a low-power state. The system will automatically return to a working state when it receives some sort of request, and will do so in a very short amount of time compared to what it would take to boot the system.

In each of the suspend states, the system stops performing all work and is either placed into either a special low-power state supported by the platform, or it is completely shut off. These special low-power platform states keep power supplied to some components, allowing certain events (a key press or lid open) to generate a hardware interrupt and cause the system to power on. Most of the inoperable system states require hardware support, because they also depend on power being supplied to memory, but there is at least one that doesn't require hardware support.

Common Name	ACPI Name
Fully On	S0
Standby	S1
Unused	S2
Suspend-to-RAM	S3
Suspend-to-Disk	S4

Table 2: ACPI System Power States

Regardless, the exact state of the CPU and every device in the system is saved when the system is suspended and later restored when it regains power. This allows the system to continue on from exactly the point at which it left off.

ACPI enumerates the different system suspend states for supported platforms, though the names are not meaningful for any other platform. It also provides an abstract interface for entering and leaving the suspend states, which provides a similar level of ease as its interface for entering CPU idle states. As is the case with those, platforms that do not implement ACPI require that that coordination be done manually. A list of ACPI system states are defined in Table 2.

The most common suspend state is suspend-to-disk. With this state, the contents of memory will be written to unused disk space before the system is powered down. When the system regains power, the contents of memory are read from the disk and restored. This is the one suspend state that does not require hardware support, though it can leverage it for generating wakeup events.

When the system is powered on, the kernel begins a normal boot sequence before it detects whether or not there is a saved memory image on the disk. If it discovers an image, it reads and reloads it into memory. This happens regardless of whether the physical state the hardware was in was a special low-power state.

Linux supports suspend-to-disk with the swsusp (swap suspend) implementation. With this code, the saved memory image is written to unused swap space. It has matured rapidly in the recent past and is currently supported on x86, x86-64, and PowerPC platforms [swsusp].

There are two alternative efforts that improve upon swsusp. Suspend2 includes several rewritten components of swsusp, several fixes to improve stability, and a few user-friendly features, like a graphical progress screen. This implementation exists as an external patch, though it is still actively maintained [Suspend 2]. swsusp3 is an implementation that moves the components to save and restore memory on disk to userspace. This reduces the amount of kernel code significantly and allows for easy integration of manipulative tasks to the saved memory image (e.g. compression and encryption). swsusp3 is currently in an alpha state, though it is likely to evolve quickly [swsusp3].

Beyond suspend-to-disk, there is one other state that is commonly found on many platforms: suspend-to-RAM. This is a special hardware state in which most or all of the components in a computer are powered off, except for RAM, which is put into a self-refresh state to preserve its contents. Before entering this state, the kernel saves the state of every device in the system, including the CPU, by copying it into memory. When the system regains power, each device is reinitialized and the state is restored.

Suspend-to-RAM provides an additional challenge in its handling of devices. On suspend-to-disk, the system goes through a boot sequence that will initialize devices to a usable state. On x86 platforms, this means that the BIOS will setup any devices that it is responsible for (video). This is not the case during suspend-to-RAM. Control is transferred to the kernel before any reinitialization is done, leaving the burden solely on the shoulders of

the kernel. In order to provide correct operation, every device driver that can be used on a system that supports suspend-to-RAM must be modified (which turns out to be an overwhelming majority of drivers). On top of that, some drivers do not have the ability to reprogram the devices that they normally support because the initialization sequences are kept proprietary (e.g. video devices again).

There is one more relatively well-known suspend state called "standby" in some literature that deserves an honorable mention. It has the ability to put the system into an inoperable low-power state, but retain the context of all devices if necessary. (Other states often remove power from the buses, implicitly removing power from downstream devices and causing their state to be lost.) However, standby is seldom used in that form, and though it is technically supported by Linux, its implementation offers no latency benefits over suspend-to-RAM.

All power states can be entered by using the sysfs PM interface. The `state` file returns the system states that are supported by the platform. Writing one of those state names to the file will cause the system to transition to that state.

4 Platform Power Management

The maturity and flexibility of Linux make it possible to port the kernel to nearly any imaginable computer. (Sometimes it seems like it has already been done [linuxdevices.com].) Each of those computers has the potential to conserve power in some way using the hardware features described above, though the combination of features used and the policy to guide them depends on several platform-specific characteristics, such as:

- What the device is used for, e.g. communications, engineering, gaming, etc.
- What the end goal of increased efficiency is, e.g. longer battery life, quieter operation, etc.
- How much tolerance there is for performance and latency, and how it is measured.
- What the physical constraints of the comptuer are, and what the risks are of having a thermal failure are, besides simply damaging the components.

In reality, there can be could any arbitrary number of conflicting design goals and requirements, spanning a nearly infinite range of computers running Linux. However, for the sake of discussion, the design goals have been narrowed to those above, and the range of computers has been narrowed to 4 broad classes of devices and 8 categories within each. Table 3 describes these categories, along with examples from each. The following sections provide an analysis of each category with the criteria above to illustrate the power management potential in Linux.

4.1 Embedded Devices

The term "embedded" has become a catch-all phrase meaning roughly any computer that is built with the intent of running a single workload and usually not very configurable or extensible. Definitions may vary, and exceptions are rampant, but here it has been narrowed to three categories of systems: handhelds, consumer electronics, and embedded controllers.

In a basic sense, each is considered an appliance by most of its consumers. They may not know or care that it is running Linux, and have

Comptuer Class	Common CPUs	Category	Products	Example
Embedded	arm	Handheld	Mobile Phone	Motorola
	omap		PDA	Nokia 770
	xscale		Media Player	iRiver h320
	mips	Consumer Electronics	LCD Television	Dell LCD
	x86		Game console	Sony PlayStation 3
	ppc		PVR	Tivo
	arm		Wireless router	LinkSys WRT54GL
Commodity	x86	Laptop	Ultra Mobile	
General Purpose	ppc		Mobile	
			Portable Desktop	
		Desktop	Gaming	
			Surfing	
			Publishing	
Professional	x86	Workstation	Engineering	
General Purpose	ppc		Graphics	
		Small Server	File server	
			Web server	
			Mail Server	
Enterprise	x86	High Availability	Infrastructure	
	ia64		Telecom	
	ppc			
	sparc	Collaborative	Distributed Application	
	mips		Database	

Table 3: Classes and Categories of systems supported by Linux

no intent to make a general-purpose computer or run their own custom kernel on it. As such, these platforms must behave like other appliances in that category by being as easy-to-use and causing as few problems as is commonly expected by comparable alternatives.

4.2 Handhelds

Handheld devices are portable devices that can easily fit into a person's hand or pocket, such as mobile phones, PDAs, and media players. The primary power management goal is to maximize the amount of battery life of these devices, though a number of constraints prevent that.

These devices are evolving rapidly as the CPUs that are being used (arm, omap, xscale) are experiencing rapid advances in performance capabilities. To gain competitive advantage over their competitors, OEMs are adding more features, which imposes greater constraints on the battery life and the ability to manage it. Mobile phones are getting media players and higher quality cameras. PDAs are getting more applications and more wireless technologies. Media players are getting higher in quality and also obtaining new wireless technologies.

In order to maximize battery life, an aggressive power management scheme must be employed. Devices that are not being used must be transitioned to a low-power state as soon as possible (e.g., when a device stops playing video/audio). The increase in latency to get the device back to a usable state is tolerable, as it is common for extended features of these platforms.

These devices may always be "on" or in a state ready to respond to user input or an incoming call, so they can not use any type of inoperable

system state. However, they are ideal candidates for operable system states, and the concept of operating points (as defined by DPM) is typically used. A CPU in one of these systems can be scaled to a fraction of its peak voltage while leaving other devices active and waiting for input. On receiving input, the system is transitioned to a higher power state, allowing it to perform the necessary work reasonably fast.

4.3 Consumer Electronics

Though handhelds could easily fit into this category, this division is made to isolate devices that are used within the home, perform a specific appliance-like function, and can assume a steady current from a wall outlet. These devices include things LCD televisions, other media consoles (video games, personal video recorders), and wireless routers. Competition is fierce for these devices, and is typically waged around aesthetics and value (number of features for the price), so power consumption is not a primary concern in implementing them.

However, that does not obviate the need for it. The impetus for power management in consumer electronics is to reduce the power bill of the consumer, since a savvy consumer may easily have a half-dozen such devices; and to reduce thermal output, since a thermal overload could result in serious damage to the consumer's residence or self.

These devices are expected to perform consistently fast whenever they are on. A slow response time or a fluctuation in response time will annoy the consumer and discourage them from purchasing that brand in the future. When these devices are not being used, they can be completely turned off.

By using sufficiently low-power devices, little power management is necessary. But the market demands that more features be added, so the components are getting richer and faster, meaning that the power consumption will continue to increase.

Operable low-power system states can be used in the future to manage power, but with the caveat that the system can never run in a state lower than one which reasonably guarantees a satisfactory response time. This should not be a challenge, since current systems perform the basic functions at a reasonable rate with little or no power management. As features are added and device speeds increase (as well as their efficiency), the devices required to use those features should remain at their lowest possible state until the feature is used by the consumer. The features should require a known amount of performance, so the device performance needed for them should be adjusted up to perform the work reasonably well.

4.4 Commodity General Purpose Computers

Commodity general-purpose computers provide the pathology to power management. Consumers of these systems want the best of both worlds—the most performance and the least power consumption. This would not be an outrageous goal if the working set of platforms and devices needing support was a reasonable size (or at least bounded). There is no such luck in the universe and therefore we have a very large set of variables in the power management formulas for commodity systems.

Fortunately, by being the mainstream, these systems and their power management features have enjoyed the most collective exposure by developers, so the problems are at least understood, even if the solutions are not. Most of the Linux power management code is designed for systems of this nature, so progress is well under way in this area.

Even though laptops and desktops can be used for nearly any task, they are typically used for doing only one subset of things at a time. Even if every possible application is executing at once, the user only has the ability to do a few things simultaneously (e.g. write a document, listen to music, and chat over an instant messaging client).

Based on what is being done, intelligent decisions can be made about how to regulate the power consumed. First and foremost, operable system states can be implemented to define boundaries on the power states of device components. By specifying which state, or "profile" to be used, the user can dictate how aggressively the power should be managed. Observed behavior has shown that the difference between each operable state is likely to be that some devices will be on, others will be off, and the performance of the CPU will vary based on an algorithm specific for that state. Depending on what the primary objective of a state is, the frequency modulation should exhibit different behavior when scaling the frequency down (aggressively or conservatively) and when scaling the frequency (aggressively or conservatively). An aggressive downward algorithm combined with a conservative upward algorithm will provide a system that stays at a low-power state unless absolutely necessary. This would benefit a lightly loaded system, as in one that was only editing documents and listening to music.

The inverse (conservatively downward and aggressively upward) will produce a system that is operating at or near its peak at all times. This will benefit systems running resource-intensive applications, like games.

Regardless of the operable state specified by the user, the system must always be able to apply the proper power management policy. The average person will not remember to always set the ideal operable state for the program they are running, so the system must either compensate with flexibility in policy or enter the proper state for the application running.

System suspend states can be used aggressively when these devices are not being used. If the system falls idle, then the user is typically not in front of it, and if they are not in front of it, they typically don't expect to use it until they sit back down, in which case they can expect a reasonable latency to return the system to a working state.

4.5 Professional General Purpose Systems

Professional general purpose systems are not a far derivation from commodity general purpose systems, but they are distinguished here to illustrate the difference in workloads and expectations. They are divided into two categories: workstations, on which people typically perform some type of engineering work; and small servers, on which departments and small companies run their infrastructure.

Workstations are high performance machines that are expected to operate at their maximum potential when they are being used, which is usually only when a user is at the keyboard in front of it. Even if there is not an application currently executing, it can be assumed that their will be one soon. And, when they do execute, they must complete the task as soon as possible. It is possible to leverage some amount of operable state system power management, though probably only with very conservative downward algorithms and very aggressive upward algorithms.

When a workstation is not directly being used, it may still be expected to be usable (i.e. by remote login), so even though it may experience long periods of idleness, it may never be able to enter an inoperable suspend state. Instead,

the operable performance can be scaled down very aggressively so that it consumes a minimum amount of power while it is ready and waiting.

Small servers typically start out as general purpose computers, but then become dedicated to running one task all the time, like a database server, a web server, or a file server. In many cases, there is no type of power management that is feasible for a single server—they must always be responsive to external requests, and they must provide a low-latency response to those requests.

Depending on the usage and the actual demand for the system's resources, some power can be managed with operable system power states. A system must be able to execute and respond at a rate that is acceptable to its users. If the components are much faster than is needed or expected, the speed of some of the components may be scaled down without sacrificing the user expectations. Additionally, depending on the usage, a server may experience very different usage models during different parts of the day. By analyzing the usage over time, different operable states can be used during different hours to conserve power but still provide the necessary availability.

4.6 Enterprise Computing

The 'enterprise' is a place where big computers live on a pathological scale. Its relationship to power management is no different. Besides the workstations, departmental servers, laptops, and handheld communication equipment, it also contains a set of servers in a class of their own. These systems, multi-machine versions of each "small server," as well as network infrastructure servers (dhcp, dns, authentication), communication servers, and distributed applications.

The performance of these systems is expected to be consistently good. They can not endure any amount of inoperability, and any increase in latency is usually unacceptable, even the relatively small latency of transitioning a CPU from a low-power C state to the C0 state (~1 ms).

However, these systems provide great opportunities for power management. These systems are large and there may be dozens or hundreds of nodes on the same problem. They require a lot of power to operate, which means they need a lot of space and a lot of cooling. If a computer runs at its maximum speed for an expected lifespan of three years, the cost to supply power to the computer will equal the initial cost of the computer. By conserving a fraction of the power consumed, an organization may save a significant amount of money in doing so.

Implementing power management on a cluster of systems is largely outside the scope of this paper. However, there is a simple analogue between operable system power states and "operable cluster power states" where the cluster performs at a rate less than its peak, but still does an acceptable amount of work in a reasonable amount of time. Under periods of decreased load, individual computers can be powered down, or they can be put into a lower-power operable state. The states to use, when to use them, and how to measure their success is a function of the application and the usage of it and is left as an exercise to the reader.

Conclusion

The concept of regulating power consumption has existed for decades in popular rhetoric about conserving natural resources. Given their finite nature, the current usage models, and shortage of mitigation techniques, most studies

suggest that current resources will inevitably depleted. Many people agree that it's important to be conscious of this.

Power management has proliferated in the computer industries over the last decade because of the economic potentials that it offers. By making devices more efficient, a company can gain a market advantage over its competitors. By using more efficient computers, a company can reduce its operational overhead. And, by applying more efficient manufacturing processes over time, higher-performance components can be used under tighter power constraints, opening up new usages and new markets for the company. Consumers realize many benefits from power management. They get longer battery life, lower power bills, and continuously increasing performance of their computers.

In fact, the only downside to power management is that the rapid parallel evolution of hardware intelligence, power management features, and user expectations imposes a stiff requirement on the software management of each. This is especially true in Linux—the kernel supports many different architectures, several of those architectures can be used in many types of computers, and many devices can be used on any platform.

This paper has provided a survey of power management concepts, how those concepts are supported by Linux, and how they are—or could be—applied to all of the different categories of machines that Linux supports. The goal of this paper was to provide insight about power management and its manifestations in the hope that it will help someone implementing some type power management support in the future understand it better.

References

[watt] W. Thomas Griffith *The Physics of Everyday Phenomena, Second Edition*, 1998

[Pentium M] Intel Corporation *Intel Pentium M Processor on 90 nm Process with 2-MB L2 Cache Datasheet*, January 2006 `http://download.intel.com/design/mobile/datashts/30218908.pdf`

[cpufreq] *The Linux cpufreq subsystem and documentation* `http://www.kernel.org/pub/linux/utils/kernel/cpufreq/cpufreq.html`

[ACPI] HP, Intel, Microsoft, Phoenix, Toshiba, *Advanced Configuration and Power Interface Specification, Revision 3.0a*, December 30, 2005 `http://acpi.info/DOWNLOADS/ACPIspec30.pdf`

[DPM WP] IBM and MontaVista Software *Dynamic Power for Embedded Systems*, Version 1.1, November 19, 2002 `http://www.research.ibm.com/arl/projects/papers/DPM_V1.1.pdf`

[DPM Project] *Dynamic Power Management Project* `http://dynamicpower.sourceforge.net/`

[PCIe PM] Intel Corporation *The Emergence of PCI Express in the Next Generation of Mobile Platforms*, Second-Generation Intel Centrino Mobile Technology, Volume 09, Issue 01, February 17, 2005 `http://www.intel.com/technology/itj/2005/volume09issue01/art02_pcix_mobile/p04_power_management.htm`

[linuxdevices.com] *The Linux Devices Showcase*, http://linuxdevices.com/articles/AT4936596231.html

[ACPI Docs] Len Brown, et al. *ACPI4Linux Documentation Overview*, http://acpi.sourceforge.net/documentation/index.html

[swsusp] Pavel Machek, et al. *Linux Swap Suspend Implementation*, Linux kernel v2.6.16, kernel/power/swsusp.c

[Suspend 2] Nigel Cunningham, et al. *Suspend 2 for Linux* http://www.suspend2.net

[swsusp3] Rafael J. Wysocki, et al. *Linux Swap Suspend Implementation*, Linux kernel v2.6.16, http://lists.osdl.org/pipermail/linux-pm/2006-January/001770.html