Linux Kernel Improvement: Toward Dynamic Power Management of Beowulf Clusters

Fengping Hu, Jeffrey J. Evans

Adaptive Computing Systems Laboratory
Purdue University
West Lafayette, IN 47907
{fhu, jje}@purdue.edu

Abstract

As a cost-effective alternative to large supercomputers, Beowulf clusters are now deployed worldwide to support scientific computing. While Beowulf clusters yield high computing performance, they also pose several challenges: 1) Heat induced hardware failure makes large scale commodity clusters fail quite frequently, 2) Powering all the computer nodes without adapting to the varying work load makes Beowulf cluster less cost-effective. In this paper, we address these issues by introducing Dynamic Power Management (DPM) into the Beowulf clusters. DPM of a Beowulf cluster means dynamically reconfiguring the cluster nodes so that node will be enabled quickly when requested and be placed into a low power state quickly to save power when the service is completed. DPM of the cluster relies on the availability of power management technologies on each computer node. This paper evaluates the current power management technologies available in computers and presents a Linux kernel improvement that makes Advanced Configuration and Power Interface (ACPI) exploitable in a cluster environment. A simple fixed timeout DPM algorithm is also evaluated on a 128 node Beowulf cluster.

1 Introduction

Beowulf clusters are scalable performance clusters based on commodity hardware and a private system network, generally using an open source systems software (Linux) infrastructure. Beowulf systems are now deployed worldwide, chiefly in support of scientific computing. While Beowulf clusters yield high computing performance, a variety of other issues have also emerged. Cluster users and administrators have become more cognizant of the fact that large scale commodity clusters fail quite frequently [1]. The main source of the cluster failure is heat-induced hardware failure. As processors grow faster and clusters scale larger, the situation is becoming more acute. The overheating-induced cluster failure problem is also complicated by the fact that Beowulf clusters are sometimes deployed in an under-cooled cluster room (data center).

Two distinct approaches can be used to address this problem: (1) Data center cooling techniques, and (2) Power management techniques. The most common approach of cooling is to install extra cooling capacity. However, the increasingly high power density of modern cluster makes this approach reach its practical limits. Sizing the air conditioning and intuitive distribution of air is no longer appropriate for high power density data centers. More sophisticated methods of cooling technologies include: (1) Using Computational Fluid Dynamics (CFD) modeling to optimize the layout of computer equipment and the cooling resources [2], and (2), dynamic cooling control in a data center environment [3], which utilizes a distributed sensor network to measure local conditions of a data center and distribute cooling resources locally.

Another approach is Power Management (PM) techniques. PM was initially developed to reduce energy consumption when a machine is not in active use. There are two evolution stages of power management in Personal Computers (PCs). Advanced Power Management (APM) represents the first stage, developed by Intel and Microsoft in 1992. APM allows the BIOS to perform power management by reducing the CPU speed, turning off the display, or disabling the hard disk after a preset period of idle time. APM has two major drawbacks: (1) Power managing is done in the background by the BIOS instead of Operating System (OS), and (2), the APM BIOS is specific to the hardware platform and has to be supplied

by the manufacturer. Advanced Configuration and Power Interface (ACPI) represents the second stage of development. ACPI is an open industry specification establishing standard interfaces for OS-directed configuration and power management of laptops, desktops, and servers. ACPI is superior to APM in that it addresses the two drawbacks of APM.

Despite of the fact that ACPI specification v1.0 was released in 1996, there has been little work addressing using this technique cluster wide. One reason is that Linux's support for ACPI is still evolving. However, if remaining operational and behavioral hurdles can be overcome, power management technologies for a Beowulf cluster can not only increase system reliability by reducing heat dissipation resulting from lower power consumption but also results in savings in electricity cost from energy and cooling.

Beowulf nodes can be thought of as a CPU and memory package which can be plugged into the cluster, just like a CPU or memory module can be plugged into a motherboard. An ideally power managed Beowulf cluster should dynamically reconfigure the cluster nodes so that any node will be enabled quickly when requested and be placed into a low power state quickly to save power when the service is completed. We call this Dynamic Power Management (DPM) of the Beowulf cluster. ACPI, as a system level power management technique made it possible to control the power states of individual computer nodes. To conserve energy while remaining quickly available, ACPI-compatible node may enter one of several system sleep states. The ACPI specification defines five of these states, known as S-states. No work is done by any of the system under S-states. Each state introduces greater power savings but requires commensurately more time to awaken and begin performing work.

Implementing DPM on a cluster relies on the availability of power management technologies on each computer node. ACPI sleep states in Linux are still evolving and they are best considered "experimental" [4]. Effort is still needed to improve this. This paper presents our evaluation of S-states in which a discrepency was discovered in the Linux kernel. Our improvement enables centralized control of power states by the cluster head nodes. The paper is organized as follows. Section two lists some related work. Section three presents the states of ACPI development on Linux. Section four explains how ACPI S-states are made exploitable for DPM of a cluster. Section five analyzes Fixed Timeout DPM algorithm. Section six concludes and discusses future work.

2 Related Work

Hsu, C. and W. Feng [5] proposed a power-management algorithm called beta-adaptation that addresses heat-related reliability for processors by controlling their clock speeds in a performance-aware manner. Dynamic Voltage and Frequency Scaling (DVFS) support of the CPU is required for this algorithm to work. The algorithm is based on the observation that a processor often times cannot execute instructions at its maximum clock rate due to performance bottlenecks in the memory, I/O, or network subsystems; thus, its clock speed can be reduced to save power without having a major effect on the overall speed of the computation. The author concluded that their preliminary experimental work demonstrates that their approach can easily be applied to commodity processors and can reduce heat generation by 30% on average with minimal effect on performance when running the SPEC benchmarks.

Benini, L., A. Bogliolo, and G. De Micheli [6] analyzed Dynamic Power Management (DPM) as a System-Level power management design methodology. DPM dynamically reconfigures systems to provide the requested services and performance levels with a minimum number of active components or a minimum load on such components. DPM encompasses a set of techniques that achieves energy-efficient computation by selectively turning off (or reducing the performance of) system components when they are idle (or partially unexploited). DPM is used in various electronic designs.

3 ACPI in the Linux Kernel

Though ACPI specification was released in December 1996, Linux's support for ACPI is still an on-going developing process and has not been widely applied to clusters. The project called ACPI4Linux on sourceforge.net aims at integrating ACPI into the Linux kernel. A development group formed around a number of developers from Intel has already written a "generic" UNIX implementation of the ACPI subsystem and is now integrating it with Linux's internal device management.

3.1 ACPI Specification and ACPI CA

The ACPI [7] specification is an open industry standard first released in December 1996. It was developed by Hewlett-Packard (HP), Intel, Microsoft, Phoenix and Toshiba. The ACPI specification defines common interfaces for hardware recognition, motherboard and device configuration and power management. "ACPI is the key element in Operating System-directed configuration and Power Management (OSPM)" [7]. ACPI has two major improvements as a standard in power management. First, it puts the OS in control of power management. The previous APM model assigns power management control to the BIOS, with only limited intervention from the OS. On the other hand, in ACPI, the OS has nearly complete control over the power management while the BIOS only provides the OS with methods for directly controlling the details of the hardware. The other important feature of ACPI is that it enables new power management technologies to evolve independently in operating systems and hardware while ensuring they continue to work together. ACPI brings power management features previously only available in portable computers to desktop computers and servers. An ACPI-compatible system may be put into extremely low consumption states, in which only memory or not even memory is powered; but ordinary interrupts (keyboard, modem, network interface card, etc.) can quickly wake the system.

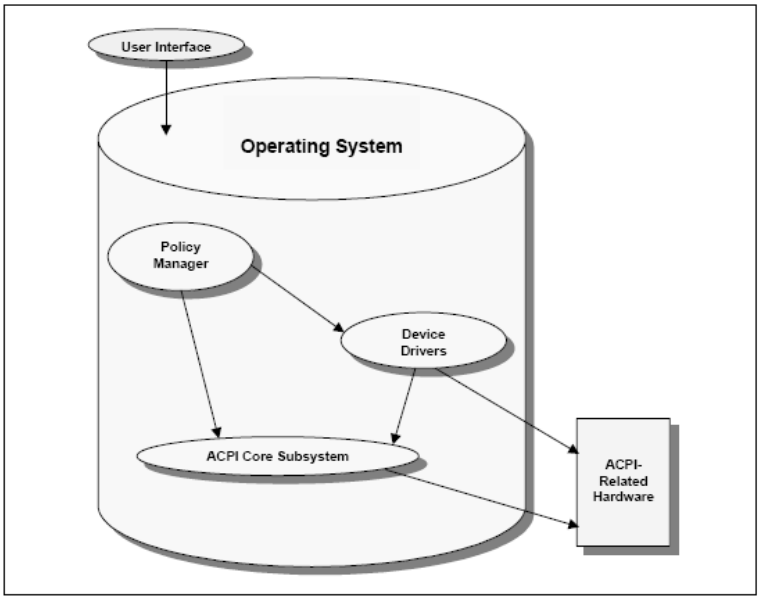

Figure 1: ACPI Component Architecture. [8]

Intel developed the Advanced Configuration & Power Interface Component Architecture (ACPI CA). It provides a reference implementation of software components that enable OS support for interfaces defined in the ACPI specification. As Figure 1 shows, it includes the following components [8]: (1) A user interface to the power management and configuration features, (2) A power management and power policy component (OSPM), (3) A configuration management component, (4) ACPI-related device drivers (i.e. drivers for the Embedded Controller, Smart Battery, and Control Method Battery), (5) An ACPI Core Subsystem component that provides the fundamental ACPI services (such as the AML interpreter), and (6), An OS Services Layer for each host operating system.

3.2 ACPI Implementation Overview

Figure 2 shows the implementation architecture. The ACPI kernel components centered around the ACPI CA core includes: OS Services Layer (OSL), linux/acpi, /proc/acpi, optional "acpi drivers". These components are corresponding to the components listed in precious section. OSL provides the conversion layer that interfaces the ACPI Core Subsystem to the Linux OS. "linux/acpi" represents the Linux-specific ACPI code such as boot-time configuration. "/proc/acpi" implements the virtual file system that will provide the user interface. Optional "acpi drivers", such as Processor and Battery are loadable modules that implant policy related to those specific features and devices.

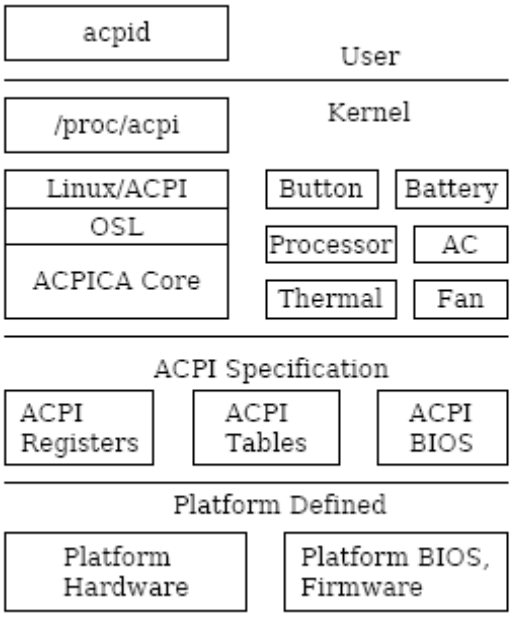

Figure 2: Implementation Architecture. [4]

4 ACPI S-states for DPM on Beowulf Clusters

From a user-visible level, an ACPI system can be thought of as being in one of the states in the Figure 3. In a typical use scenario, a computer alternates between the working and sleeping states. The computer is used to do work only in the working state. In the working state, individual devices can be in low-power (Dx) states and processors can be in low-power (Cx) states if they are not being used. Through tuning within the working state, trade-offs among speed, power, heat, and noise can be achieved. The sleeping states are entered when the computer is idle or when the user issues the state transition command. No user-visible computation is done in a sleeping state. The main difference between sleeping states is the amount of time it takes to wake up and the amount of power reduction achieved. The five sleeping states defined by ACPI specification are:

- The S1 is the most power-hungry among the sleep modes. All processor caches are flushed and the processors stop executing instructions. The processor-complex context and dynamic RAM context is maintained, thus power to the CPU(s) and RAM is maintained. Devices that do not indicate they must remain on may be powered down. This state is not supported by some newer machines; but some older machines are more likely to support S1 rather than S3.
- State S2 is logically lower than the S1 and it conserves more power. In the S2 state, the CPU is powered off, thus it requires the operating software to flush all dirty cache to dynamic RAM. S2 is not commonly implemented.
- The S3 state is logically lower than the S2 and it conserves more power (S3 is also known as Suspend to RAM; it is called Standby in Windows and Sleep in Mac OS X). In this state, dynamic RAM is virtually the only component that is powered. It is functionally the same as the S2 state from the software viewpoint except that some more components may be powered down in S3. S3 is widely implemented.
- The S4 is logically lower than the S3 and it conserves more power (S4 is also known as Suspend to disk; it is called Hibernation in Windows, Safe Sleep in Mac OS X). In this state, dynamic RAM context is not maintained. All content of dynamic RAM is saved to a hard drive, preserving the state of the operating system, applications etc. S4 is quite different from other S1-S3; a computer in S4 state is able to resume even if power loss occurred when a computer is in S4 state.

- The S5 (G2) is similar to G3 Mechanical Off (Shown in Figure 3). The boot procedure must be run to bring the system from G2 to G0 Working. The difference between S5 and S4 is that S4 enables the user to resume work where it was left off.

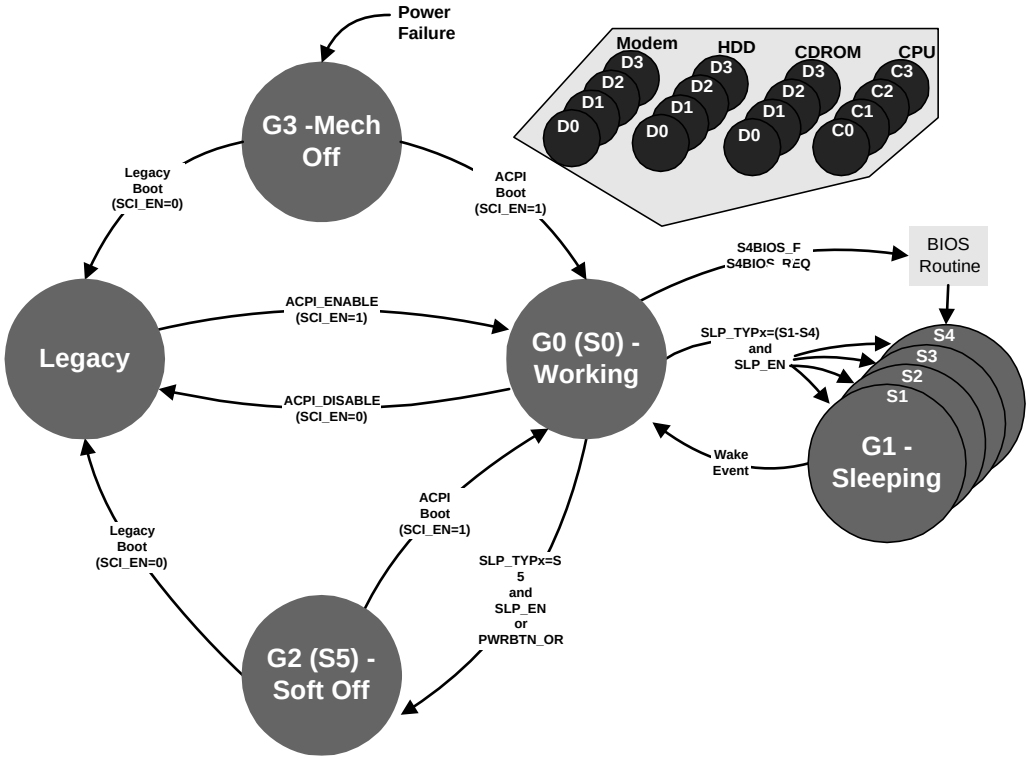

Figure 3: ACPI Global System Power States and Transitions. [7]

4.1 Centralized Control of Cluster Node Power States

In order to use ACPI S-states for DPM of a cluster, the power states of individual cluster nodes must be controlled by the cluster head. ACPI compatible computer systems may be put into ACPI S-states by issuing a command through the OS interface. However, a computer in an ACPI S-state is no longer operable (i.e., no instructions can be executed by a CPU in a sleep state). The system must be enabled through hardware interrupts. The interrupt can come from the keyboard, modem, network interface card, etc. Since cluster nodes are interconnected through their network interface cards, an existing technology called Wake-on-LAN (WOL) can be used to generate this interrupt [9]. WOL technology was first released by International Business Machines Corporation (IBM) Advanced Manageability Alliance in April of 1997. Other industry initiatives, such as the Intel-based Wired for Management, soon supported the WOL standard. A WOL compatible network adapter examines all data packets without concern to the protocols seen and searches for a “Magic Packet” [10], which consists of a head of six 0xFF followed by the physical address of the card(48-bit MAC address) repeated eight times. The ACPI specification requires the OS to ensure any bridges between the waking device and the core logic are in the lowest power state in which they can still forward the wake signal. Upon receiving the valid “Magic Packet”, the network adapter sends the defined signal on its bus. Bus bridges forward this signal to upstream bridges using appropriate signal for that bus. Thus eventually, the signal reaches the core chip set (e.g.: an ACPI chip set), which in turn wakes the machine.

4.2 **Addressing the General Purpose Event (GPE) Storm Problem**

All testing, subsequent operating system kernel modifications and verification studies were conducted on the 128-node Linux cluster in the Adaptive Computing Systems Laboratory (ACSL) in the College of Technology at Purdue University. The compute nodes are identical 933MHz Pentium III class machines, equipped with 512MB of RAM and 40GB of disk space. An additional head node similar to the compute nodes (but with 250GB of disk space) is used for user login and system control. The head node does not participate in computation. There is also a machine that mirrors the head node, primarily for data backup purposes.

We began our implementation of system-wide DPM by first testing the state transition matrix of the ACPI (figure 3). The System Control Interrupt (SCI) is registered to OS by the ACPI. It is the only entrance for all ACPI interrupts. One of the two event types signaled by the SCI is called a General Purpose Event (GPE). An important use of GPE is to implement device wake events. When using WOL as the stimulus we observed all the attempts instructing nodes to transition from sleeping states S1, and S3 to S0 failed (state S2 is not supported by our cluster nodes). On the other hand, the transition from S5 to S0 was functional and reliable. These results motivated further investigation deeper into the kernel software. Here it was noticed that the wake GPE is disabled within the function "acpi_leave_sleep_state" located in hwsleep.c. It was then determined that the GPE was not being disabled by the interrupt dispatcher (SCI handler), resulting in a behavior that continually executed the interrupt, subsequently never allowing the higher level function “acpi_leave_sleep_state” to execute. This behavior had been given the name of the "GPE storm problem" by Linux kernel developers.

The ACPI specification states that the SCI status bit corresponding to the waking device should be cleared once the operating system is up and running. The run time GPEs are enabled at run time while the "wake" GPE should be disabled at run time. The wake GPE should be enabled only as the device or system is transitioning from an operational state to a sleep state. We modified the function “acpi_ev_gpe_dispatch” located in Evgpe.c to disable the wake GPEs at run time. Collaboration with the responsible ACPI developer through bugzilla (http://bugzilla.kernel.org/show_bug.cgi?id=6881) confirmed the location of the bug and that our solution was appropriate. The fix has been incorporated into ACPICA versions as of 20060831.

To verify our proposed solution we retested the state transition matrix in figure 3. A cluster node is directed to enter a sleep state by locally executing the command "`echo x > /proc/acpi/sleep`", where `x` is the desired sleep state (e.g. S1). To re-awaken the node we execute the WOL tool (`wakeonlan`) from the cluster head node, which in turn broadcasts the magic packet to the desired cluster node. Our repeated tests on each of the 128 compute nodes on our cluster yielded encouraging, but not perfect results. As shown in table 1, the S5 to S0 transition was reliable both before and after our fix, indicating that the fix did no harm. In addition, S1 transition demonstrates reliable operation after our solution was implemented. The transition from sleep state S3 to S0 continues to exhibit some systemic instability. Here it was observed that a subset of the 128 compute nodes do not transition from S3 to S0, this is complicated however in that the subset of nodes failing to enter state S0 changed on each attempt. We are actively investigating this behavior to determine its root cause.

Table 1: WOL Failure Rate.

Sleep State	Before Modification	After Modification
S1	100%	0%
S3	100%	5-10%
S5	0%	0%

In order for an S-state to be used in DPM of Beowulf clusters, the S-state has to be reliable (i.e. hundred percent successful state transition between working state and S-state). The test showed the S1 and S5 satisfy this condition; however the S3 is till not a reliable state. According to Brown, A.L [4], the power management side of ACPI still needs plenty of bug-fixes. The most visible will probably be anything that makes Suspend/Resume work on more platforms [4]. Sleep states are unreliable on Linux because Suspend/Resume suffers from (at least) two systematic problems [4]:

- Items initialized in “_init()” and “_initdata()” may be referenced after boot (i.e. during resume).
- PCI configuration space for both devices and PCI bridges is not uniformly saved and restored.

5 Fixed Timeout DPM Algorithm

Consider a system which has instantaneous transition time between power states: performing power state transitions does not introduce power or performance costs. In such a system, implementing a DPM is straightforward. An optimal policy for such a system would be to transition the system to and from the deepest sleep state as soon as the system is idle or a request arrives respectively. In reality, performing the power state transition of a cluster node has non-negligible power and performance cost. Several design techniques for system level power management are discussed in [6]. Equations 1-4 [6] are used to calculate the power saving resulted from an optimal DPM policy. T_{BE} (equation 3) is defined as the minimum inactivity time required to compensate the cost of entering the sleeping state. In general T_{BE} is the sum of two terms: the total transition time (T_{TR}, equation 1) and the minimum time that has to be spent in the low-power state to compensate the additional transition power (P_{TR}, equation 2). An optimal policy for systems with non-negligible power and performance cost would shut down the cluster node at the beginning of all idle periods longer than T_{BE} and wake it up in time to serve the upcoming requests so that a minimum delay is introduced. For such a system, the power saving is given by equation (4), where $T^{avg}_{idle>T_{BE,S}}$ is the average length of idle periods that are longer than $T_{BE,S}$, F is the cumulative probability distribution function of T_{idle}. This optimal policy gives the upper bound of the power savings that can be achieved by a DPM policy.

$$T_{TR} = T_{ON-OFF} + T_{OFF-ON} \quad (1)$$

$$P_{TR} = \frac{T_{ON-OFF} * P_{ON-OFF} + T_{OFF-ON} * P_{OFF-ON}}{T_{TR}} \quad (2)$$

$$T_{BE} = T_{TR} + T_{TR}\frac{P_{TR} - P_{ON}}{P_{ON} - P_{OFF}} \quad (3)$$

$$P_{saved,S} = (P_{ON} - P_S)\frac{(T^{avg}_{idle>T_{BE,S}} - T_{BE,S})}{T^{AVG}_{idle}}(1 - F(T_{BE})) \quad (4)$$

Table 2 (approximate data) gives the ACPI S-states power consumption and state transition time based on our compute nodes. As can be seen from table 2, S3 and S5 are the most profitable states (achieves the greatest power savings). Since S3 also has the lower break-even time than S5. S3 seems to be the best sleep state for DPM of a Beowulf cluster in this case.

Table 2: Cluster Node State Transition Table.

Power saving state	P_{OFF} (W)	P_{ON-OFF} (W)	T_{ON-OFF} (S)	P_{OFF-ON} (W)	T_{OFF-ON} (S)	T_{BE} (S)
Idle(S0)	30 W	-	-	-	-	-
Standby(S1)	23 W	30	1	35	2	6
Suspend to ram(S3)	3 W	30	2	50	5	10
Soft Off(S5)	3 W	35	20	45	50	100

In most real-world systems, the system cannot determine whether an idle period will be longer than T_{BE} at the beginning of that idle period nor can it know when the next job request will arrive. This makes power management a power optimization problem under performance constraints, or vice versa [6].

Existing techniques for DPM and policy optimization includes predictive techniques and stochastic control [6]. One of the most widely used DPM policies is fixed timeout. Under the fixed timeout policy, the system is put to sleep after the idle period reaches T_{TO} (Fixed timeout threshold). Interestingly, T_{TO} has the same effect to the actual power savings as T_{BE} because both represent the portion of idle time that can not be effectively exploited to save power. Based on equation (4), the power saving can then be written as equation (5) by neglecting the transition power cost. This equation shows that the expected energy saving is a function of T_{TO} and the distribution of T_{idle}. Assume T_{idle} has exponential distribution (based on the assumption that job submission is a Poisson process). The energy saving can then be written as equation (6), where λ is the job arrival rate.

$$E_{saved,S}^{avg} = (P_{ON} - P_S)(T_{idle>T_{TO}}^{avg} - T_{TO})(1 - F(T_{TO})) \quad (5)$$

$$E_{saved,S}^{avg} = (P_{ON} - P_S)(T_{idle>T_{TO}}^{avg} - T_{TO})e^{-\lambda T_{T0}} \quad (6)$$

From the above analysis, we can see the potential power savings that can be achieved through a simple fixed timeout algorithm. When this power saving of individual cluster nodes is multiplied by the total cluster nodes number in a Beowulf cluster, a huge amount of power savings can be achieved. This power saving will also lower the heat dissipation and thus reduce the over-heat induced hardware failure problem of a Beowulf cluster.

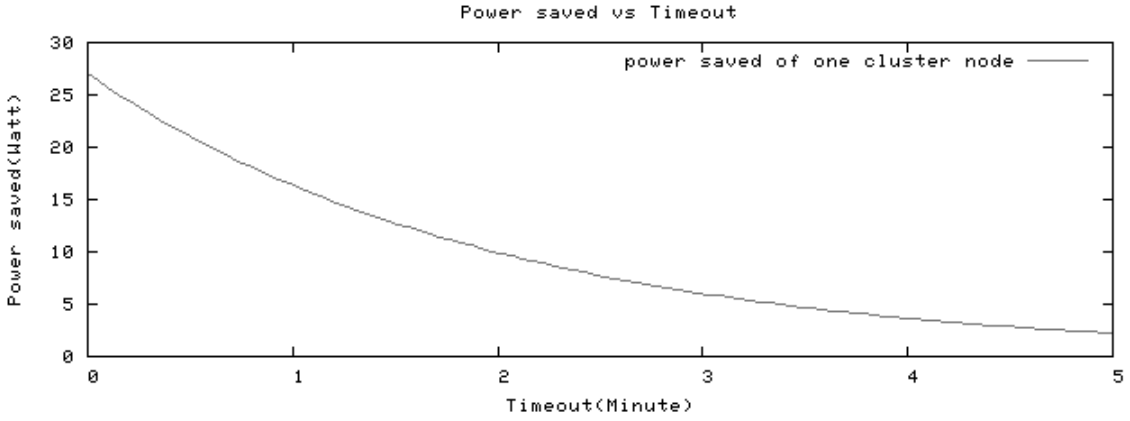

Figure 4: Power Saved vs Timeout.

Our Linux Kernel improvement provides reliable WOL performance from ACPI sleep states S1 and S5. This enables implementing DPM on Beowulf clusters. A simple fixed timeout policy has been implemented on the ACSL cluster. Under this DPM policy, cluster nodes are transitioned to S5 state after 30 minutes of idleness; nodes are transitioned to S0 (working) on the arrival of a request. This simple DPM policy has showed significant power savings at the expense of a few minutes of job execution startup latency.

6 Conclusion and Future Work

DPM technologies have been studied and used in all types of electronic systems. This paper introduces DPM into Beowulf cluster by addressing some of the problems with ACPI on Linux that prevent DPM from being implemented on a cluster. DPM technology for a Beowulf cluster is promising in that it can increase system life by reducing heat dissipation resulting from lower power consumption. Reducing cluster power consumption also results in reduced operational cost in electricity for energy and cooling.

Our implemented Kernel improvement provides reliable ACPI sleep state transition between S1, S5 and S0. However, as Table 1 illustrates, State S3 seems to be the most profitable in that its power saving is

almost as big as S5 and the latency introduced is almost as small as S1, however state transition reliability remains a concern. Thus more work is required to realize an exploitable S3 sleep state transition to S0.

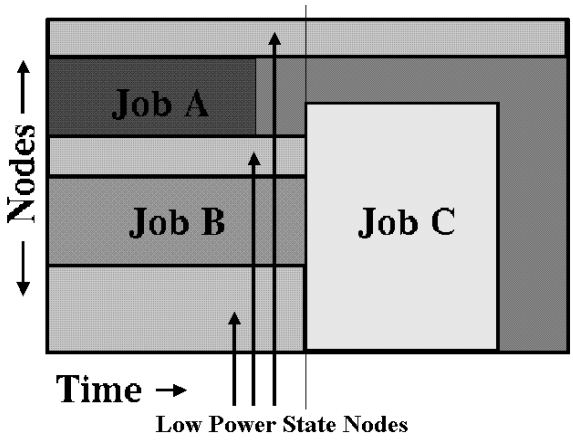

Figure 5: Scheduler Awareness of DPM in Beowulf Cluster.

We are developing more advanced DPM algorithms in order to optimize power saving and system performance. DPM of a Beowulf cluster can take advantage of using job scheduler knowledge. Figure 5 illustrates scheduler awareness of DPM on a cluster: Job A and B is in a running state, at the same time job C is waiting in the job queue. This illustrates an example of system where we do have knowledge about a request before it really arrives. Under such a condition, those nodes that are not occupied by job A and B can be put into a low power state to save the energy. When job B is ending, some of the sleeping nodes should be enabled and Job C can be executed in time.

Acknowledgements

This work was supported by College of Technology, Rosen Center for Advanced Computing (RCAC), and Cyber Center at Purdue University. The authors also wish to thank the reviewers for their helpful suggestions to improve the presentation and technical content of this paper.

References

1. Hsu, C. and W. Feng. *Reducing Overheating-Induced Failures via Performance-Aware CPU Power Management*. in *The 6th International Conference on Linux Clusters: The HPC Revolution 2005*. 2005.
2. Patel, C.D., et al. *Computational fluid dynamics modeling of high compute density data centers to assure system inlet air specifications*. in *IPACK'01--The PacificRim/ASME International Electronics Packaging Technical Conference and Exhibition*. 2001. Kauai, Hawaii.
3. Boucher, T.D., et al. *Viability of dynamic cooling control in a data center environment*. in *Thermal and Thermomechanical Phenomena in Electronic Systems, 2004. ITHERM '04. The Ninth Intersociety Conference on*. 2004.
4. Brown, A.L. *The State of ACPI in the Linux Kernel*. Proceedings of the Linux Symposium 2004 [cited 2006 Dec 7]; Available from: www.linuxsymposium.org/proceedings/reprints/Reprint-Brown-OLS2004.pdf.
5. Hsu, C. and W. Feng. *A Power-Aware Run-Time System for High-Performance Computing*. in *SC|05: The ACM/IEEE International Conference on High-Performance Computing, Networking, and Storage*. 2005. Seattle, WA

6. Benini, L., A. Bogliolo, and G. De Micheli, *A survey of design techniques for system-level dynamic power management.* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2000. 8(3): p. 299-316.
7. Intel, et al. *Advanced Configuration and Power Interface Specification*. 2006 [cited 2006 Dec 07]; Available from: http://www.acpi.info/DOWNLOADS/ACPIspec30b.pdf.
8. Intel. *ACPI Component Architecture Programmer Reference*. 1999 [cited 2006 Dec 10]; Available from: http://developer.intel.com/technology/iapc/acpi/downloads/ACPICA-ProgRef.pdf.
9. Lieberman, P. *White Paper: Wake on LAN Technology*. 2002 [cited 2006 Dec 10]; Available from: http://www.liebsoft.com/index.cfm/whitepapers/Wake_On_LAN.
10. AMD. *Magic Packet Technology*. 1995 [cited 2006 11/07]; Available from: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/20213.pdf.