Operating-System Directed Power Reduction

Yung-Hsiang Lu, [†]*Luca Benini, Giovanni De Micheli*
CSL, Stanford University, USA. {luyung, nanni}@stanford.edu
[†]DEIS, Università di Bologna, Italy. lbenini@deis.unibo.it

Abstract

This paper presents a new approach for power reduction by taking a global, software-centric view. It analyzes the sources of power consumption: tasks that require services from hardware components. When a component is not used by any task, it can enter a sleeping state to save power. Operating systems (OS) have detailed information about tasks; therefore, OS is the best place for power reduction. Our technique is effective in identifying hardware idleness and shutting down unused components. We implement this technique in Linux and show that it can save more than 50% power compared to traditional hardware-centric shutdown techniques.

1. Introduction

Low-power design is increasingly important because of the popularity of portable devices and the concern of environmental impact of electronic systems [16]. Due to rapid advance in hardware, modern systems are supporting wide ranges of applications. Specifically, computers and some PDA's deploy operating systems, such as Windows, Palm OS and Linux to manage resources. This paper focuses on such systems.

In many systems, some hardware components (also called *devices*) are not always used; workload variations make adaptive power reduction possible. These techniques reduce power consumption according to the requests generated from running tasks. If no task requires a particular device, the device is *idle* and can be *shut down* to enter a low-power low-performance *sleeping state* (also called *stand-by state*). When a running task requires this device, it is *woken up* and enters a high-power high-performance *working state*. Dynamically determining power states according to workloads is called *dynamic power management* (DPM) [3]. *Power managers* (PM) determine power state transitions according to their shutdown rules (also called *policies*).

ISLPED '00, Rapallo, Italy.

Power management can be generalized for more than two power states [5]; each state has different performance and power consumption. Therefore, power management includes dynamic voltage setting [7] and variable clock speeds [14]. Setting voltages or clock speeds is equivalent to choosing power states. We use "power management" for all techniques that dynamically reduce power based on workloads. For simplicity of the following presentation, we consider only two power states: working and sleeping.

Power management in commercial products is mainly based on timeout— shutting down a device when it has been idle long enough. Most research activities on DPM take a hardware-centric view; they observe past requests at the target device to predict future idleness [3] [8] [11]. Some schemes use stochastic models due to the lack of information to distinguish requesters [4] [17] [13]. None of these approaches makes distinction on the request source; they implicitly assume that there exists one requester. In many systems, however, requests may be generated by multiple requesters. For example, hard disk IO's may be generated by a compiler, a text editor, or a file transfer program (`ftp`). Similarly, network transmission requests can come from an Internet browser or a `telnet` session.

Our main contribution is to introduce a power reduction technique in operating systems (OS) with an accurate system-level model of requesters: concurrently running tasks. Two tasks are concurrent if one starts before the other terminates. Detailed information about tasks is available in OS kernel. Running tasks may terminate and new tasks may be created. After a task terminates, it will not generate new requests. Classifying requests according to their sources provides more information to power managers about future requests. We call this approach *task-based power management* (TBPM). We implement it in Linux on a single-processor computer for controlling the power states of a hard disk drive (HDD) and a network interface card (NIC). Experiments show that our approach saves more than 50% power compared to traditional power reduction techniques.

2. OS-Directed Power Management

OS-directed power management (OSPM) can be divided into two categories: adjusting CPU clock

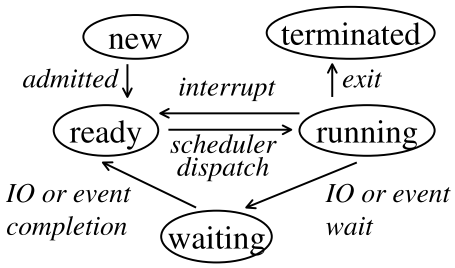

Figure 1: process states

speed [7] [6] [10] [14] [18], or putting devices into sleeping states [3] [4] [8] [13]. This paper focuses on the second problem.

In 1997, Advanced Configuration and Power Interface (ACPI) [1] was proposed as a standard protocol between OS and hardware for saving power. Based on ACPI, Microsoft's *OnNow* [12] and *ACPI4Linux* [2] support power management in operating systems.

All power management schemes proposed so far are based solely on the observation of requests at the target device; they do not exploit extra knowledge available from the OS. We call this hardware-centric approach *device-level power management* (DLPM). At the device level, information about individual requesters is unavailable. Therefore, all requests are assumed to come from a "black-box" requester. DLPM has three major drawbacks. First, different tasks can have substantially different request patterns; DLPM cannot make decisions based on task-specific patterns. Second, tasks can be created and terminated. If a device is used by only one task, it can be shut down immediately after the task terminates. DLPM usually needs to wait unnecessarily even though no requester exists. Third, some tasks may have tight performance requirements; DLPM has no information to meet task-specific performance constraints.

In contrast, TBPM is a software-centric approach; it uses the knowledge available in operating systems for power management by dividing requests according to tasks, which correspond to OS *processes*. A process can be in one of several states such as running and ready, as shown in Figure 1 [15]. The terms *process*, *requester* and *task* are used interchangeably in this article.

Figure 2 is an example when TBPM outperforms the timeout approach. Suppose there are two processes using a network transmitter: `telnet` and `ftp` before

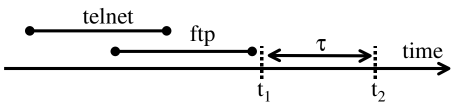

Figure 2: example when TBPM outperforms timeout

	r_1	r_2	r_3	r_4	
d_1	$U_{1,1}$	$U_{1,2}$	$U_{1,3}$	$U_{1,4}$	.....
d_2	$U_{2,1}$	$U_{2,2}$	$U_{2,3}$	$U_{2,4}$	.....
d_3	$U_{3,1}$	$U_{3,2}$	$U_{3,3}$	$U_{3,4}$	.....

	gcc	emacs	netscape
HDD	12	0.7	0.4
NIC	0	0	2.3

Figure 3: utilization matrix (top) and an example

t_1. No process uses the transmitter afterwards; consequently, it can sleep at t_1 to save power. However, timeout-based DPM waits for τ (the timeout value), and keeps the device in the working state for τ unnecessarily.

3. Task-Based Power Management

TBPM has better requester models because it can deal with four problems that device drivers cannot handle:

1. Requests are generated by multiple requesters (tasks). TBPM uses the knowledge from OS kernel to separate tasks.
2. Tasks are created, executed, and terminated. Device drivers have no knowledge of the existence of multiple tasks and their termination.
3. Tasks have different characteristics in device utilization. For example, a compiler and a text editor have different request patterns.
4. A task can generate requests only when it is in the running state; namely, it occupies CPU time. TBPM considers the CPU time of tasks while deciding power state changes.

3.1. Data Structures

TBPM uses a two-dimensional data structure called the *device-requester utilization matrix*, $\mathbf{U}$. Additionally, it creates a vector called the *processor utilization vector*, $\mathbf{P}$. The following paragraphs describe these data structures, the update rules, and how to shut down devices.

Matrix $\mathbf{U}$ stores the relation between devices and requesters. $U(d, r)$ is an element of $\mathbf{U}$; it represents the utilization of device d by requester r. Figure 3 shows a utilization matrix. The number of devices is system-dependent; the number of requesters is unlimited. When a new requester is created, one column is allocated; when a requester terminates, its column is deleted. This figure shows an example of two devices (HDD and NIC) and three requesters (gcc, emacs, and netscape). Since gcc and emacs do not send any network packets, their utilization for NIC is zero. $\mathbf{P}$ is the processor utilization for each process; $P(r)$ represents the percentage of processor time used by requester r.

3.2. Updating U, P

An element of matrix $\mathbf{U}$, $U(d,r)$, is defined as the reciprocal of the average *time between requests* (TBR) for device d by requester r. In practice, it is not advisable to compute TBR as the running average of all times between requests since the beginning of a task. A discounted average, where the immediate past is weighted more than the remote past, is a more effective way to estimate TBR [8]. $U(d,r)$ and TBR after n requests are obtained by

$$\begin{aligned} &TBR_n = a \cdot TBR + (1-a) \cdot TBR_{n-1} \\ &U(d,r) = \tfrac{1}{TBR_n} \end{aligned} \tag{1}$$

where TBR is the interval between the last two requests and $0 < a < 1$ is the discount factor. When $a = 0$, TBR_n is a constant using the first TBR; when $a = 1$, only the last TBR is used. Experiments show that a value of a between 0.2 and 0.8 produces satisfactory results while 0.5 is suggested in [8].

$P(r)$ is the percentage of CPU time executing task r, or

$$P(r) = \frac{CPUTime(r)}{\sum\limits_{\forall \text{ requester } \rho} CPUTime(\rho)} \tag{2}$$

$\mathbf{P}$ is updated based on a sliding window scheme, instead of discounted average, for two reasons. First, when a process is IO-bounded, generating bursty requests, TBR will be dominated by how fast a device can serve requests. While this properly shows that the process has high device utilization, it does not capture the percentage of running time this process takes. Second, when a process generates bursty IO requests, it stays in the *running* state (Figure 1) only momentarily each time it is selected by the OS scheduler. Discounted average cannot obtain an appropriate estimation of the percentage of CPU time taken by this process. Consequently, a sliding window is used to compute the CPU time distributed among processes. The window size should be large enough to sample the execution of all processes and short enough to quickly reflect workload variations. Figure 4 shows relative power consumption of a hard disk for different window sizes (the experimental setup will be described in Section 4); the power is normalized to one for a window size of five minutes. We use two minutes as the window size for a balance between adaptation speed and accuracy.

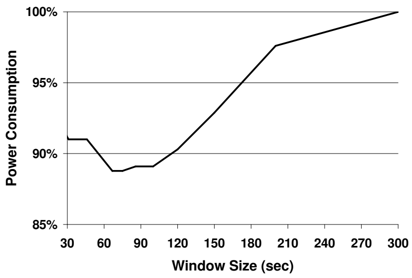

Figure 4: power and window sizes (100% for 300 sec)

3.3. Shutdown Condition

The key decision made by the power manager is when to shut down a device. Changing power states takes time and extra energy. A quantitative measure of shutdown cost is the *break-even time*, T_{be} [3]. T_{be} is the minimum length of an idle period to amortize the power cost of shutting down a device. T_{be} is a device characteristic independent of workloads. Let P_w and P_s be the power consumption in the working and the sleeping states. T_{sd} and T_{wu} are the time required to shut down and wake up the device. E_{sd} and E_{wu} are the energy for shutdown and wakeup. T_{be} can be obtained by this formula: $P_w \cdot T_{be} = E_{sd} + E_{wu} + P_s \cdot (T_{be} - T_{sd} - T_{wu})$ or:

$$T_{be} = \frac{E_{sd} + E_{wu} - P_s(T_{sd} + T_{wu})}{P_w - P_s} \tag{3}$$

TBPM shuts down a device d if its total utilization, $U(d)$, becomes smaller than a threshold; that is, when the following condition is true

$$U(d) = \sum_{\forall \text{ requester } r} U(d,r) \times P(r) < th \tag{4}$$

$U(d)$ is the request rate for device d created by all tasks. To compute total utilization, $U(d,r)$ is multiplied $P(r)$, because a task generates requests only when it is running on the CPU.

We choose th to be k/T_{be}. When $k = 1$, the shutdown condition has an intuitive interpretation. If the overall request rate is larger than $1/T_{be}$, the expected time between requests is smaller than T_{be} and a shutdown is not beneficial. In practice, k may be different from one. A larger k makes the policy more aggressive since a device is shut down even though its utilization is still high. Because of our performance rule (3.4), we can choose k larger than one without significantly degrading performance. Figure 5 shows the relative power consumption of a hard disk for different k values; the power for $k = 1$ is 100%. It shows that the power consumption is larger for smaller k; when $k \gg 1$, the performance rule will

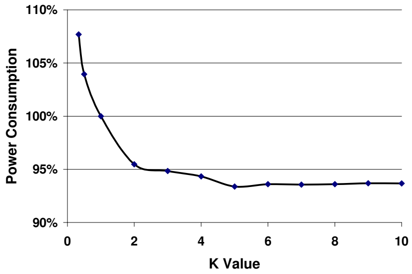

Figure 5: power and k values (100 % for k = 1)

prevent TBPM from shutting down a device too often and leads to the same power saving. We choose 2 for k in our experiments.

3.4. Performance Consideration

TBPM is designed to save power without compromising performance. We target interactive systems such as personal computers. The performance metrics are related to user-perceived interactivity; interactivity is affected by waiting. While power managers cannot change hardware parameters such as T_{sd} and T_{wu}, they can determine how often user-perceived waiting occurs. If a PM issues many shutdown commands within a short time period and dramatically increases response time, users will perceive a drastic degradation in interactivity. Furthermore, the users may react to obtain response and cause a steep increase in system loads. This "positive feedback" due to multiple shutdowns within a short period must be avoided. Previously proposed power management policies focused mainly on power saving without providing any performance guarantee in worst-case behavior [11].

Our approach guarantees that no more than two consecutive shutdowns are issued within T_w. Performance guarantee is achieved by preventing shutting down a device if a shutdown was issued less than T_w seconds ago. Because TBPM is capable of detecting long idle periods more accurately, this rule has only negligible effect on power.

3.5. TBPM Procedure

Figure 6 pictorially illustrates TBPM. This figure shows how power management is integrated into process management compared to Figure 1. A requester column is allocated whenever a new task is created; the column is deleted when the task terminates. Utilization is initially set to zero; then it is updated whenever a request is issued. The power manager evaluates the utilization in the process scheduler. The total device utilization is the sum of utilization weighted by CPU time of each requester.

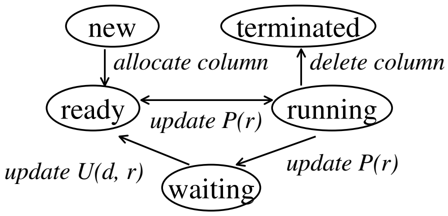

Figure 6: updating $U(d,r)$ and $P(r)$

TBPM behaves consistently under heavy or light workloads. For a lightly-loaded system running the idle process mostly, sparse requests will not cause the power manager to keep a device in the working state too long since $P(r)$ is small for this requester. When the system has heavy workloads but it does not use a device frequently, the power manager will shut down the device soon after it is used since $U(d)$ is small. When the system is running tasks that heavily use a device, TBPM will keep the device in the working state. Therefore, TBPM is robust in all cases.

4. Implementation and Experiments

4.1. Implementation Platform

We conduct our experiment on a personal computer. We implement TBPM in Linux kernel V 2.2.5, Redhat distribution 6.0, to control the power states of a hard disk and a network transmitter. We focus on transmission because (1) a transmitter consumes much more power than a receiver, particularly in wireless communication (2) network receivers have to listen to incoming signals and cannot be completely shut down.

We modify the kernel and related device drivers in Linux. It is a fully functional computer with X Window and networking; we use it for daily tasks such as developing TBPM code, editing texts, and surfing the Internet. This computer is configured as a typical client. Server daemons, such as `http` server and Internet news server are turned off; `cron` tasks are scheduled at lower frequencies. Power state changes in an HDD and an NIC are emulated with two states: one working state and one sleeping state.

For evaluating the effectiveness of our approach and comparing it with other power management policies, we emulate power-state changes without actually setting the hardware power states. The implementation maintains a set of variables that record when a particular policy shuts down a device and when the device is woken up. Therefore, we can run multiple policies for identical workloads generated by real users. These variables are recorded every ten minutes on a second hard disk so data collection does not interfere with normal operation.

Table 1 shows the parameters of the devices in our experiments. They are typical for commercial devices: a 2.5" mobile hard disk (such as Fujitsu 2043AT) and a wireless Ethernet card (such as WaveLAN 8484411481).

4.2. Power Management Overhead

Column allocation for new processes in the utilization matrix, $\mathbf{U}$, can be implemented in a fixed-size circular queue to avoid the overhead of dynamic memory allocation. The size should be large enough to accommodate all concurrent processes. We use a 64-column matrix because there are usually fewer than 64 processes on a personal computer. When more processes are running, columns are assigned by the least-recently-used (LRU) rule [15]. The processor utilization vector, $\mathbf{P}$, also contains 64 elements. In total, less than 160 KB memory is used; it is allocated at the unswappable memory region to avoid extra disk IO's. There is no human-perceivable performance degradation.

4.3. Experimental Results

We compare TBPM with four device-level power management policies (1) exponential regression relationship between two adjacent idle periods [8] (2) event-driven semi-Markov model [17] (3) 2-competitive policy which sets the timeout value (τ) to T_{be} [9] (4) timeout with one and two minutes (τ = 60, 120). One minute is the minimum unit in many user-controlled power management settings, such the Control Panel in Microsoft Windows. Table 2 compares these four policies on a typical workday for ten hours. T_w is thirty seconds for the performance guarantee in TBPM.

In this table, T_s is the time in the sleeping state (sec); N_d is the number of shutdowns; S_l is the length of the longest sequence that causes delay every thirty seconds (value of T_w) or shorter; P_a is the average power (W); R is the power consumption relative to TBPM. While most power management researchers concentrate on the average power (P_a) only, we are also concerned about the number of state changes (N_d) because each state change has associated overhead.

Device	P_w	P_s	T_{wu}	T_{sd}	E_{wu}	E_{sd}
HDD	0.95	0.13	1.61	0.63	4.39	0.36
NIC	1.43	0.05	0.24	0.10	0.50	0.05

Table 1: hardware parameters. unit: Watt, sec and J

Device	Policy	T_s	N_d	S_l	P_a	R
HDD	TBPM	23641	181	0	0.435	1.00
	[8]	17856	325	2	0.586	1.35
	[17]	22828	581	9	0.507	1.16
	[9]	22552	477	3	0.499	1.15
	τ = 60	16551	101	0	0.586	1.35
	τ = 120	12347	64	0	0.677	1.56
NIC	TBPM	29121	179	0	0.316	1.00
	[8]	22411	361	3	0.576	1.82
	[17]	27155	597	11	0.398	1.26
	[9]	28492	457	8	0.345	1.09
	τ = 60	20789	99	0	0.640	2.00
	τ = 120	16257	48	0	0.808	2.55

Table 2: policy comparison

Three policies ([8] and τ = 60, 120) have much higher average power consumption (P_a). The "pre-wakeup" scheme in [8] wakes up a device before a request arrives; therefore, T_s is much smaller. Also, 60 and 120 seconds are too long to determine idleness. They waste power in the first 60 (or 120) seconds of an idle period. The other policies (TBPM, [9] and [17]) consume less power. Among them, TBPM has smaller numbers of shutdowns because TBPM can find long idle periods more accurately. The performance rule described in 3.4 prevents two or more shutdown commands within T_w seconds; consequently, users do not perceive repetitive delays in a short time period. For each policy, S_l indicates how long a user may need to wait repetitively. For example, a user needs to wait as many as 9 times for HDD within four and half minutes in [17]; each waiting can be longer than two seconds ($T_{sd} + T_{wu}$).

Although performance rules can be applied to any power management policy, the policy has to predict long idle periods with high accuracy. Otherwise, performance rules will keep devices in their working states unnecessarily and waste power. TBPM, with additional information from OS kernel, finds long idle periods more accurately. Therefore, TBPM is essential for using a performance rule to reduce power consumption without compromising user satisfaction.

4.4. Qualitative Analysis of Experimental Results

Regression relationship in [8] can cause multiple shutdowns within a short time period and frustrate users. Consider Figure 7; each block indicates a request. A long idle period, such as a lunch break, is followed by bursty requests, such as working after the lunch. Because of the long idle period, the regression policy predicts that new idle periods will also be long and shut down the device repetitively. This policy shuts down the

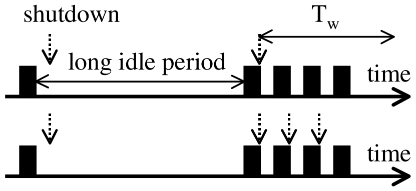

Figure 7: TBPM (top) does not cause repetitive shutdowns as [8] does. Each block indicates a request.

device for the first several idle periods and creates depressing user experience. In contrast, TBPM does not shut down any device too frequently; therefore, it provides satisfactory performance in the worst cases. It refrains from shutting down the device again within T_w after the previous shutdown. During this time, TBPM quickly updates $\mathbf{P}$ and $\mathbf{U}$ to adapt for the change of workloads. Therefore, TBPM is superior to [8] when workloads suddenly increase.

In [4] [13] [17], requests are assumed to follow certain probability distributions. TBPM, on the other hand, makes no assumption about request patterns. In fact, it is designed to handle dramatically varying patterns; TBPM learns request patterns by updating $\mathbf{P}$ and $\mathbf{U}$; hence, TBPM is more flexible.

TBPM fruitfully exploits the additional knowledge available in OS about request sources. TBPM achieves a better tradeoff between power and performance than traditional DLPM, without significant overhead in kernel. Performance penalties are tightly controlled not only in average, but also in worst-case conditions.

5. Conclusions

Traditional device-level power management assumes a single "black-box" requester even though requests may be generated by concurrently running tasks. The box is usually characterized by its statistical parameters; we claim that such information is insufficient. In this paper, we have introduced task-based power management (TBPM) that "opens the box" and classifies requests by their tasks using the information in OS kernel. TBPM computes device utilization of each task and shuts down a device when the overall utilization is low. We implement this policy in Linux and demonstrate that, when compared to device-level policy, TBPM produces better power saving, with smaller, and well-controlled performance penalty.

6. Acknowledgments

This work is supported in part by MARCO/DARPA Gigascale Silicon Research Center and in part by NSF under contract CCR-9901190.

7. References

[1] ACPI. http://www.teleport.com/~acpi.

[2] ACPI4Linux. http://phobos.fs.tum.de/acpi/index.html.

[3] L. Benini, A. Bogliolo, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Transactions on VLSI Systems*, 8(3), June 2000.

[4] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(6):813–833, June 1999.

[5] E.-Y. Chung, L. Benini, and G. D. Micheli. Dynamic power management using adaptive learning tree. In *International Conference on Computer-Aided Design*, pages 274–279, 1999.

[6] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU. In *ACM International Conference on Mobile Computing and Networking*, pages 13–25, 1995.

[7] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power Optimization of Variable Voltage Core-Based Systems. In *Design Automation Conference*, pages 176–181, 1998.

[8] C.-H. Hwang and A. C. Wu. A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation. In *International Conference on Computer-Aided Design*, pages 28–32, 1997.

[9] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive Randomized Algorithms for Nonuniform Problems. *Algorithmica*, 11(6):542–571, June 1994.

[10] J. R. Lorch and A. J. Smith. Scheduling Techniques for Reducing Processor Energy Use in MacOS. *Wireless Networks*, 3(5):311–324, 1997.

[11] Y.-H. Lu, E.-Y. Chung, T. Šimunić, L. Benini, and G. D. Micheli. Quantitative Comparison of Power Management Algorithms. In *Design Automation and Test in Europe*, pages 20–26, 2000.

[12] OnNow. http://www.microsoft.com/hwdev/onnow/.

[13] Q. Qiu, Q. Wu, and M. Pedram. Stochastic Modeling of a Power-Managed System: Construction and Optimization. In *International Symposium on Low Power Electronics and Design*, pages 194–199, 1999.

[14] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Design Automation Conference*, pages 134–139, 1999.

[15] A. Silberschatz and P. B. Galvin. *Operating System Concepts*. Addison-Wesley, 4 edition, 1994.

[16] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen. Predictive System Shutdown and Other Architecture Techniques for Energy Efficient Programmable Computation. *IEEE Transactions on VLSI Systems*, 4(1):42–55, March 1996.

[17] T. Šimunić, L. Benini, and G. D. Micheli. Event-Driven Power Management of Portable Systems. In *International Symposium on System Synthesis*, pages 18–23, 1999.

[18] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.