Comparing System-Level Power Management Policies

Yung-Hsiang Lu

Giovanni De Micheli
Stanford University

System-level power management is a trade-off among several factors, as this quantitative analysis shows.

REDUCING POWER CONSUMPTION is a challenge to system designers. Portable systems, such as laptop computers and personal digital assistants (PDAs), draw power from batteries, so reducing power consumption extends their operating times. For desktop computers or servers, high power consumption raises temperature and deteriorates performance and reliability. Soaring energy prices early last year and rising concern about the environmental impact of electronics systems further highlight the importance of low power consumption.

Power reduction techniques can be classified as static and dynamic. Static techniques, such as synthesis and compilation for low power, are applied at design time. In contrast, dynamic techniques use runtime behavior to reduce power when systems are serving light workloads or are idle.[1] These techniques are known as dynamic power management (DPM).[2] DPM can be achieved in different ways; for example, dynamic voltage scaling (DVS) changes supply voltage at runtime as a method of power management. Here, we use DPM specifically for shutting down unused I/O devices. We built an experimental environment on a laptop computer running Microsoft Windows. We implemented existing power-management policies and quantitatively compared their effects on power saving and performance degradation. A qualitative survey of power management is available in Benini et al.[2]

Power management

System-level power management saves power of subsystems (also called devices). Examples of devices include I/O controllers, hard disk drives, network interface cards, and displays. Shutting down hard disks and displays is the most widely adopted system-level power management on PCs.

Figure 1 illustrates the concept of power management. A workload consists of multiple requests. For hard disks, requests are read or write commands; for network cards, requests are packets to send or to be received. When there are requests, the device is busy; otherwise, it is idle. Here, the device is idle between T_1 and T_4. When the device is idle, it can be shut down to enter a low-power sleeping state. (The "Standby or Sleeping?" sidebar discusses how this work views these states.) In this illustration, the device is shut down at T_2 and woken up at T_4, when requests arrive again. Changing power states takes time; T_{sd} and T_{wu} are the shutdown and wake-up delays. In the example of hard disks and displays, it takes several seconds to wake up these devices. Furthermore, waking up a sleeping device may take extra energy.

In other words, changing power states has overhead. If there were no overhead, power management would be trivial: Just shut down a device whenever it is idle. Unfortunately, there is delay and/or energy overhead. Consequently, a device should sleep only if the

0740-7475/01/$10.00 © 2001 IEEE

saved energy justifies the overhead. The rules to determine whether to shut down a device are called policies. For simplicity, this article addresses only one device and one stream of requests. Because power management does not change requests, we are concerned about only the power when a device is idle in either the working or sleeping state. We don't consider the power required to serve requests.

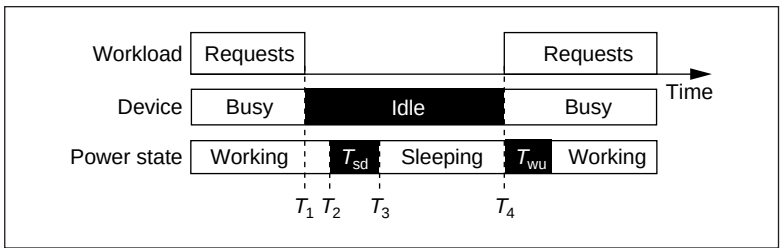


Figure 1. Power management from the workload, device, and power state points of view.

Power management degrades performance

After a device sleeps, at least one request has to wait for the wake-up delay. Some researchers propose predictive wake-up to eliminate such waiting.[3] Unfortunately, accurate prediction is difficult. Waking up too early wastes power; waking up too late does not eliminate waiting completely. Hence, this article does not discuss predictive wake-up.

Performance for interactive workloads

Many policies focus on power saving and ignore the performance impact, especially performance for interactive systems. While performance metrics for batch-mode workloads such as the SPEC[4] benchmarks have been widely accepted, there are no universally adopted metrics for interactive workloads. This is partly because of the difficulty in objectively quantifying user perception. We discuss these differences in the "Repeatable Workload" sidebar.

One simple performance metric is average delay: (the number of shutdown commands × state transition delay)/total execution time. However, when users interact with computers, they are concerned about worst-case as well as average-case waiting. Worst-case performance raises questions such as "What is the longest possible waiting time within five minutes?" or "How often do I have to wait while editing this file?" These differ from batch mode performance questions such as "How many transactions can this computer finish today?" Many users understand that occasional delays are unavoidable and tend to be forgiving if they occur infrequently. If delays occur frequently, however, user satisfaction drops dramatically. We call this phenomenon short-memory effect: Users forget delays that happened a long time

Standby or Sleeping?

Some devices have multiple low-power states. For example, some hard disks have a standby state and a sleeping state. These disks consume less power in their sleeping states compared to the standby states. However, a sleeping disk requires a hardware reset to wake up; a standby disk does not need resetting. Most power management policies assume only one working state and one sleeping state. The device can serve requests only in the working state. In this article, we make the same assumption and use sleeping for the low-power state. For hard disk drivers, this may actually refer to the standby state shown in the manufacturer's specification.

Repeatable Workload

Repeatable workloads for interactive systems are different from computation-intensive workloads, such as the SPEC benchmarks. These benchmarks issue requests continuously to keep systems at peak performance. In contrast, interactive systems are mostly idle, waiting for user inputs. User inputs are extremely unrepeatable. It is nearly impossible to ask a user to repeat exactly the same inputs at the same speed. We conquer this problem by recording user activities through a filter driver and creating a one-day trace. The trace includes all disk accesses with time stamps at 10 ms intervals.

In our experiments, we use two computers for measurement. The first computer replays the trace while one of the policies is running. Another computer detects the hard disk's power state changes on the first computer and records the time stamps of these changes. This method reduces the interference of policies and data recording on regenerating the requests from the trace.

We compute the power of a policy by (P_w × time in the working state + P_s × time in the sleeping state + E_o × the number of shutdowns) / total time. This excludes the power while the disk is reading or writing data because power management does not change requests.

ago. Figure 2 illustrates this concept. If the time between two delays is too short, users remember two consecutive delays and feel dissatisfied. When the time between two delays is longer, user memory decays and users will accept the second delay's occurrence.

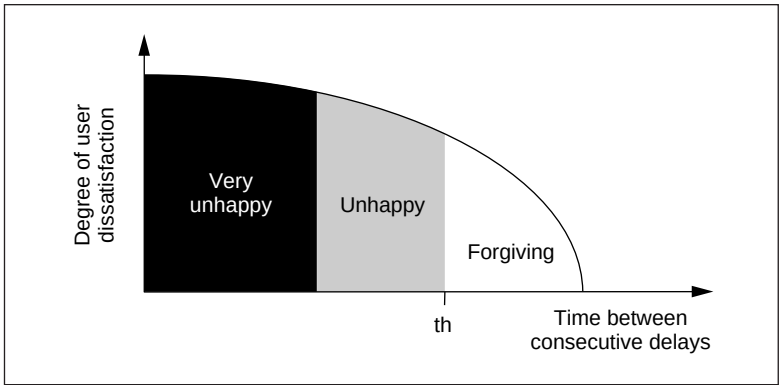


Figure 2. Relation between user dissatisfaction and delay. User memory decays with time; the longer the time between delays, the more forgiving the user.

We propose a new way to quantify performance in interactive systems. It measures the longest shutdown sequence in which the time between two adjacent shutdowns is less than threshold *th*, shown in Figure 2.

Policy classification and representative policies

Power management policies can be classified into three categories based on the methods to predict whether a device can sleep long enough. These categories are time-out, predictive, and stochastic.

Time-out

A time-out policy has a time-out value τ. Time-out policies assume that after a device is idle for τ, it will remain idle for at least T_{be}. An obvious drawback is the energy wasted during this time-out period.

Time-out-based policies include fixed-time-out, such as setting τ to three minutes. This is widely adopted in commercial products. In Microsoft Windows, users can set the time-out values for hard disks and monitors in the control panel. Alternatively, time-out values can be adjusted at runtime. For example, one adaptive time-out policy (ATO) adjusts τ by the ratio of τ and the previous idle period.[5] If the ratio is too small, this policy increases τ; if the ratio is too large, it decreases τ.

Some time-out policies consider the variations of hardware parameters. These device-dependent time-out (DDT) policies select τ based on the break-even time of the device under control. (For an explanation, see the "Break-Even Time" sidebar.) It can be proved that a policy using $\tau = T_{be}$ will consume at most twice the energy of an ideal "oracle" policy. We explain the oracle policy in the "Oracle Power Manager" sidebar.[6,7]

Predictive

These policies predict the length of an idle period before it starts, eliminating the time-out period ($T_2 - T_1$ in Figure 1).[8] If an idle period is predicted to be longer than the break-even

Break-Even Time

Power management is a prediction problem; it seeks to forecast whether an idle period will be long enough to compensate for the overhead of power state changes. The minimum length of an idle period to save power is called the *break-even time* (T_{be}).[2] It depends on individual devices and is independent of requests. Consider a device whose state transition delay is T_o (including shutdown and wake-up delays) and the transition energy is E_o. Suppose its power in the working and sleeping states is P_w and P_s. On the left of Figure A, the device is kept in the working state; on the right, the device is shut down. The break-even time makes energy consumption in both cases equal. Namely, $P_w \times T_{be} = E_o + P_s \times (T_{be} - T_o)$ or $T_{be} = (E_o - P_s \times T_o)/(P_w - P_s)$. The break-even time has to be larger than the transition delay; therefore, $T_{be} = \max[(E_o - P_s \times T_o)/(P_w - P_s), T_o]$.

It also is convenient to define the minimum sleeping time (T_{ms}) as the shortest duration in the sleeping state to save power. It excludes the transition delay: $T_{ms} = T_{be} - T_o$. In Figure 1, the device does not sleep immediately after it is idle. Its duration in the sleeping state is more important than the length of this idle period. Power management polices predict whether $T_4 - T_3 > T_{ms}$ instead of whether $T_4 - T_1 > T_{be}$.

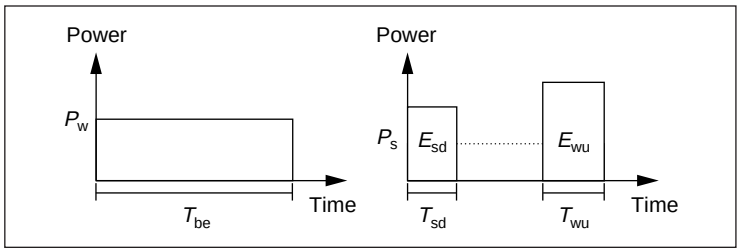


Figure A. T_{be} makes the energy consumption equal.

time, the device sleeps right after it is idle.

Requests make a device change between busy and idle. The lengths of busy and idle periods form two sequences: $B[1], B[2], \ldots$ and $I[1], I[2], \ldots$. Let $B[n]$ be the busy period before idle period $I[n]$ for any integer n. One study finds that busy and idle periods form an "L" shape.[9] Figure 3 shows our measurement of the relationship between $B[n]$ and $I[n]$; this is consistent with the finding in Srivastava.[9] Short busy periods are followed by long idle periods (left vertical stroke of the L); long busy periods are followed by short idle periods (horizontal stroke of L). Therefore, the L-shape policy shuts down a device if it is busy for only a short period of time.[9] However, this policy does not handle the situation when short busy periods are followed by short idle periods near the origin (lower left corner of L when two strokes intersect).

Some other predictive policies use only past idle periods to predict the length of future idle periods. Adaptive learning tree (LT) encodes the sequence of idle periods into tree nodes.[10] This policy predicts the length of an idle period with finite-state machines similar to multibit branch prediction in microprocessors. If an idle period is predicted longer than T_{be} and it is indeed longer than T_{be}, the confidence level increases; otherwise, the confidence level decreases. This policy uses the history encoded on the nodes and the confidence level to compute a node traversal path and to determine the power states for future idle periods.

Another predictive policy uses the predicted and the actual lengths of a previous idle period.[3] Suppose $\rho[n]$ is the prediction for $I[n]$, then $\rho[n+1] = a \times I[n] + (1 - a) \times \rho[n]$; here a is a constant between zero and one. This policy is called exponential average (EA) because $\rho[n+1]$ essentially is the average of $I[1], [2], \ldots, I[n]$ weighted by an exponential sequence if we repetitively replace ρ: $\rho[n+1] = (1-a)^{n+1} \rho[0] + a \Sigma_{i=0}^{n} (1-a)^i I[n-i]$. This policy also limits the speed of growth for ρ, $\rho[n+1] \leq c \times \rho[n]$ for a constant $c > 1$, so that the prediction is not dominated by one exceptionally long idle period.

Stochastic

Stochastic policies model the arrival of requests and device power-state changes as stochastic processes, such as Markov processes. Minimizing power consumption is a stochastic optimization problem. A simple stochastic policy models requests and power-state transitions as stationary discrete-time Markov (DM) processes.[11] At any given time, a request arrives with a probability; the device sleeps with another probability. The optimal probability for the device to sleep can be obtained by solving the stochastic optimization problem. Such a solution is valid only for geometrically distributed stationary stochastic processes. There are two ways to generalize this model: extending it to other stochastic models and including non-stationary behavior.

The first extension can be achieved by using continuous-time Markov models[12] or continuous-time semi-Markov models.[13] With these models, there is no need to evaluate the appropriate power states periodically. Instead, the

Oracle Power Manager

A device's power consumption varies widely according to different policies. A device consumes P_w if no power management is applied. On the other extreme, we can use an imaginary oracle power manager (OPM) to find the lower bound of power consumption. An OPM knows perfectly the arrival of future requests. It shuts down the device immediately after the device becomes idle if this idle period is longer than the break-even time. Hence, it achieves the best power saving. Unfortunately, an OPM does not exist in reality; perfectly predicting the future is impossible. An OPM can be simulated by analyzing a request trace offline. We use such analyses to find the lower bound of power consumption as a reference point for comparison.

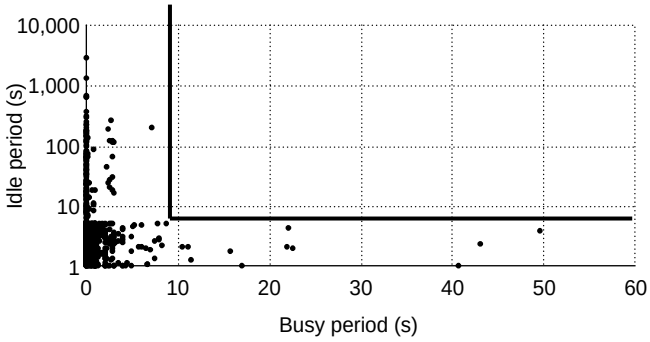


Figure 3. Relation between busy and idle periods forms an "L" shape.

arrival and service of requests are the events that trigger state transition decisions. Eliminating periodic state evaluation (as done in DM) reduces processor power consumption.

Moreover, requests and devices can be modeled by stochastic processes that are not necessarily geometrically or exponentially distributed; thus Markov processes cannot capture them. For example, a study finds that the requests for wireless communication are best modeled by Pareto distributions. This study proposes the use of time-index semi-Markov (SM) models and determines a way to compute optimal policies.[14]

The second extension considers non-stationary (NS) requests. One approach computes the optimal policies in advance for different sets of arrival rates and stores the results. The actual rates are estimated at runtime. If the estimated rate is the same as any precomputed rate, then the precomputed optimal policy is used. Otherwise, interpolation is applied between these precomputed solutions to obtain the sleeping probability.[15]

Resource requirements

Prediction accuracy and power savings are two major criteria in choosing an appropriate policy for specific applications. Resource requirements also are important, particularly for portable embedded systems in which resources are tightly limited. This section compares policies based on their resource requirements. We consider memory, computation, and timer.

Memory requirement

Some policies require one-element memory. For example, timeout policies store τs; stationary stochastic policies store the optimal shutdown probabilities. The exponential average policy stores four values: $I[n]$, $\rho[n]$, a, and c. In contrast, some polices require a significant amount of memory. NS stores optimal solutions of multiple sets of parameters. LT stores the lengths of idle periods in tree nodes. There is no explicit mechanism to limit the sizes of the trees, which can grow arbitrarily large.

Runtime computation

Time-out policies with fixed τs do not need any computation. The exponential average requires two multiplications and one addition to compute $\rho[n+1]$. ATO needs one division for the ratio of τ and the previous idle period to compute the next τ value. LT requires more computation to find an appropriate traversal path along the trees.

Timer generation

A timer is used to create an event in the future. Timers are important resources especially for time-constrained systems such as real-time systems. Timers are essential for time-out policies. When a device is idle, a timer with value τ is set. When the timer expires, the device sleeps. Timers also are used in policies with discrete-time models, such as Markov processes. In these policies, timers generate periodic events that trigger power managers to determine appropriate power states. No study has been devoted to investigating the optimal periods of these triggering timers. If the periods are too short, power managers are triggered too often and consume too many computation resources. If the periods are too long, power managers may miss power-saving opportunities. Some policies do not require timers; for example, continuous-time Markov models are event driven.[12] Similarly, adaptive learning trees do not need timers while computing traversal paths.

Implementing policies in Microsoft Windows

We built an environment to implement policies for controlling power states of the hard disk on a laptop computer running Windows 2000.[16] Our environment is based on the support of the Advanced Configuration and Power Interface (ACPI).[17] ACPI is an interface specification between hardware and software for system-level power management. ACPI allows operating systems to control the power states of hardware devices. In Microsoft Windows 2000, ACPI commands are issued by creating I/O request packets (IRP). As we write this article, Microsoft Windows are the only commercial operating systems supporting ACPI. Linux will support ACPI in kernel 2.4.

Filter driver

We built a template using filter drivers in

Windows; policies were implemented using this template. A filter driver is a device driver inserted between a Windows kernel and another device driver (or between two drivers). This is called *attaching* on the other driver. The filter driver intercepts communications between Windows and the other driver; this filter driver can pass, add, delete, or change the messages between the other driver and Windows.

Figure 4 illustrates the process of power management by a filter driver. A filter driver attaches to another driver during machine booting. This is accomplished by calling the **IoAttachDeviceToDeviceStack** function available in Windows Driver Development Kit (DDK). During initialization, the filter receives a driver object created by Windows. The filter specifies which messages to intercept by providing a filter function. For example, if the filter wants to intercept read commands, it assigns a function (**FilterReadFunc**) to the function pointer specified by the read major number (**IRP_MJ_READ**) of the driver object (**DriObj**):

```
NTSTATUS
FilterRead (IN DEVICE_OBJECT * DevObj,
     IN IRP * Irp);
DriObj->MajorFunction[IRP_MJ_READ]
     = FilterRead;
```

When Windows issues a read command, this filter function is invoked with an IRP as a parameter. The function can analyze the IRP and then pass it to the original device driver.

Change power states

A filter driver also can create a new IRP. It can change the power state of a device by creating a power IRP, for example:

```
PoRequestPowerIrp(DevObj,
IRP_MN_SET_POWER,
PowerState,
CompletionCallBack,
......);
```

This function call creates a power IRP to the device pointed by **DevObj**. The second parameter specifies that an IRP should be created to set the power state of this device. The third parameter assigns the power state for this device. ACPI has one working state (**PowerDeviceD0**) and three sleeping states (**PowerDeviceD1** to **PowerDeviceD3**). After the device enters the desired state, a call-back function specified by **CompletionCallBack** will be invoked. This allows the filter driver to perform needed tasks after the device finishes the state transition.

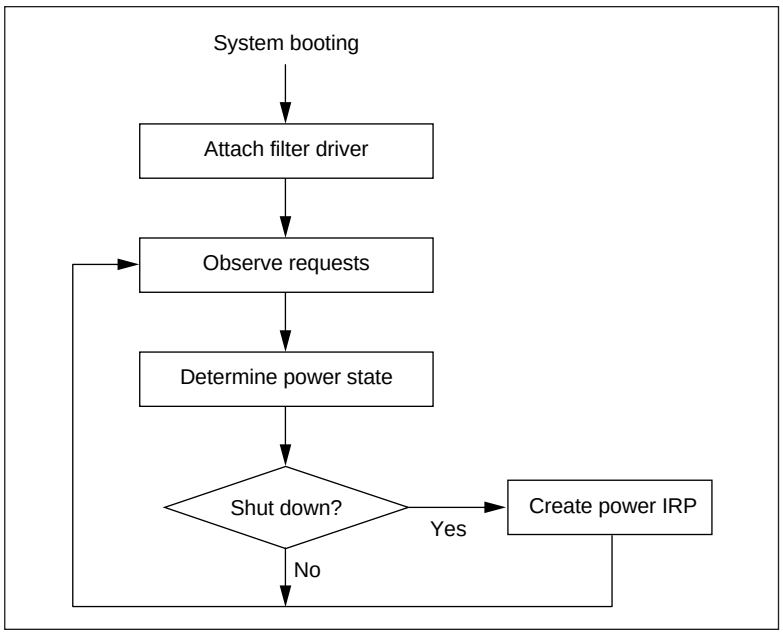


Figure 4. Power management by filter driver.

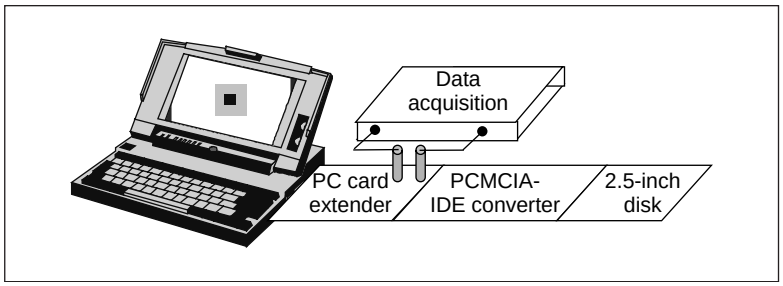


Figure 5. Setup for measuring disk power through the PCMCIA interface.

Experimental results

We conduct our experiments on a Sony laptop computer, using a test set up as shown in Figure 5. It has a primary internal hard disk and can have a secondary external hard disk through the PCMCIA (PC Memory Card International Association) interface. We insert

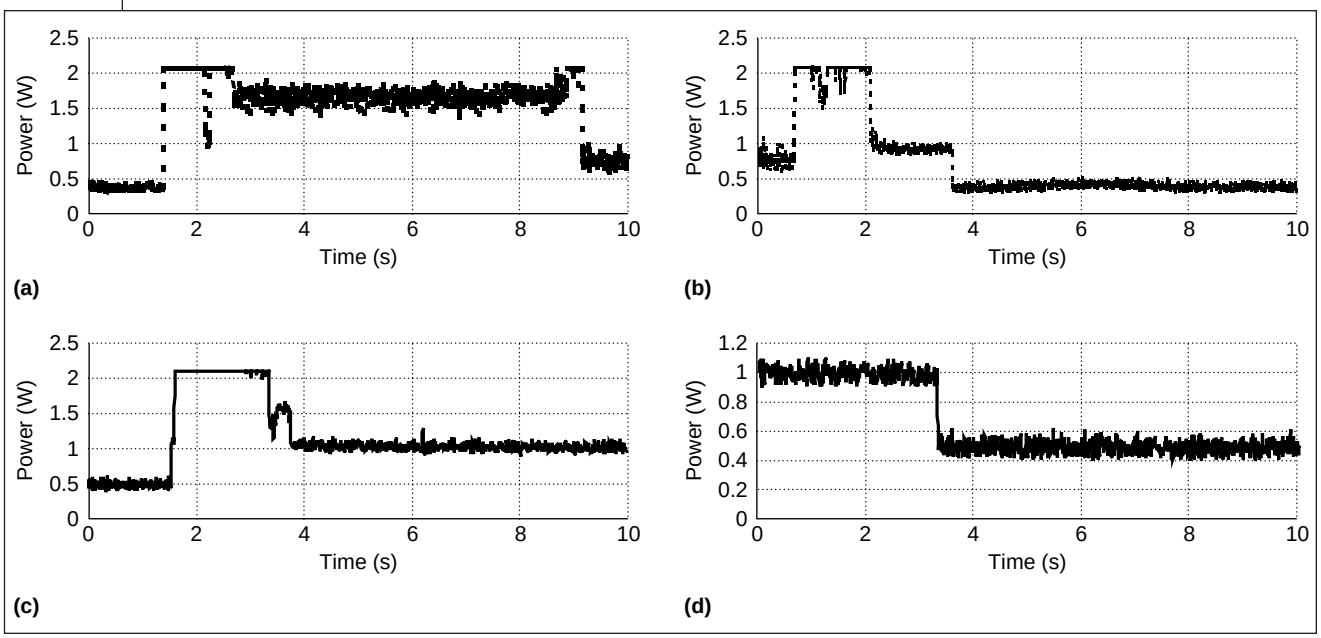


Figure 6. State transitions of two hard disks: Hitachi disk (a) wake up and (b) shut down; and Fujitsu disk (c) wake up and (d) shut down.

Table 1. Disk parameters.

Disk	P_w (watts)	P_s (watts)	E_o (joules)	T_o (s)	T_{be} (s)
DK23AA-60	0.78	0.39	17.83	10.72	35.0
MHF 2043AT	1.09	0.48	4.82	1.93	6.39

an Accurite PCMCIA Extender Card between the laptop computer and a PCMCIA-IDE (integrated drive electronics) converter and connect a standard 2.5" IDE hard disk to the computer through the converter. The extender card allows us to measure the power of a hard disk using a National Instruments data acquisition card. The laptop computer runs a beta version of Windows 2000 (NT 5 beta-2).

Device parameters

Figure 6 shows the transition between power states of two 2.5" hard disk drives. The first is a Hitachi DK23AA-60 (6 Gbyte) disk and the second is a Fujitsu MHF 2043AT (4 Gbyte) disk. This figure clearly shows that waking up a disk takes a significant amount of time and energy. Furthermore, transient power is more than twice that in the working state. Transition energy varies widely even for the same type of devices. The Hitachi disk's transition energy is more than three times that of the Fujitsu disk. Table 1 summarizes the parameters of these disks. From this point on, we discuss only the Fujitsu disk.

Figure 7 shows the power, number of shutdowns, and the percentage of incorrect shutdowns for each policy in one day. The power and the number of shutdowns are normalized with respect to the oracle power manager. The percentage of incorrect shutdowns is defined as the percentage of shutdowns that causes the disk to sleep for a period shorter than T_{ms}. Figure 8 shows the lengths of the longest shutdown sequences of four policies. In this figure, the exponential average (EA) policy has the worst interactive performance. It may cause more delays than other policies, with users experiencing delays again and again within several minutes.

Grading policies

Power saving and performance of individual policies depend on workloads. Some policies (such as EA) are not designed for controlling hard disks; some of their assumptions may not be applicable to such devices. We include such policies in our comparison to understand their

limitations. While the numbers in Figures 7 and 8 vary for different workloads and devices, the grades are valid for wider ranges because we consider their properties in addition to the numbers obtained in our experiments.

Each policy is graded by six criteria: power, number of shutdowns, shutdown accuracy, interactive performance, memory requirements, and computation requirements. We compare the first two criteria with the oracle policy.

- **Power.** If a policy consumes less than 1.15 times as much power compared to oracle, its grade is A. If it consumes 50% more power, its grade is C. Otherwise, its grade is B. A is better than B and B is better than C.
- **Number of shutdowns.** If a policy has fewer shutdowns than oracle, its grade is A. If a policy has more than 1.5 times as many shutdowns, its grade is C.
- **Shutdown accuracy.** If less than 30% of shutdown commands cause the disk to sleep a period shorter than T_{ms}, the grade is A. If more than 50% of shutdown commands cause the disk to sleep shorter than T_{ms}, the policy has grade C.
- **Interactivity performance.** We use one minute as the threshold value in Figure 2 to quantify interactivity. If the length of the longest waiting sequence for a policy is shorter than 10, its grade is A. If the sequence is longer than 20, its grade is C.
- **Memory requirements.** If a policy needs storage for only one number, its memory grade is A. If the storage is unbounded, its grade is C. The memory of a policy is bounded if the bound is explicit in the policy. For example, exponential average requires the storage of $\rho[n]$, $I[n]$, a, and c. In contrast, adaptive learning trees do not specify such a bound. At runtime, adaptive learning trees can limit the tree sizes but these limits are not specified in the policy itself. Similarly, the nonstationary policy (NS) may precompute as many sets of parameters as desired and store the results.
- **Computation requirements.** If a policy needs none or only one arithmetic operation, its grade is A. If the number of operations is unbounded, its grade is C. We consider only runtime computation regardless of any computation performed in advance.

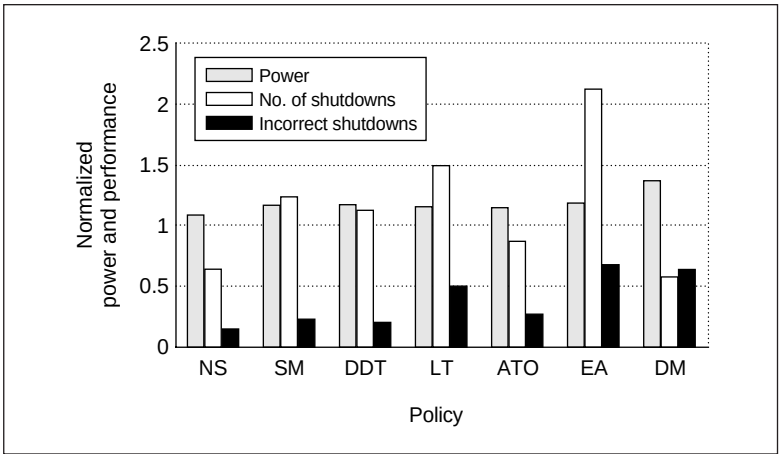


Figure 7. Power, number of shutdowns, and percentage of incorrect shutdowns for each policy. The power and number of shutdowns are normalized to the oracle policy. Shorter bars are better.

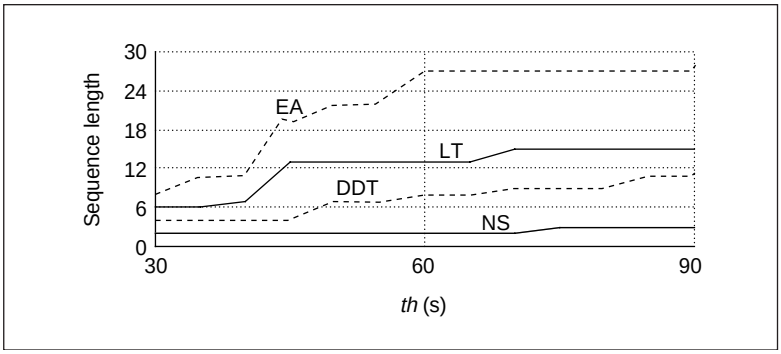


Figure 8. Lengths of the longest shutdown sequences for different *th*. Smaller numbers a preferable.

Table 2. Grading criteria for policies.

Criteria	A grade	C grade
Power	< 1.15 oracle	> 1.5 oracle
No. of shutdowns	< oracle	> 1.5 oracle
Shutdown accuracy (percentage)	> 70	< 50
Interactive performance	< 10 when *th* = 60	> 20
Memory requirements	One element	Unbounded
Computation requirements	One arithmetic	Unbounded

Table 2 summaries our grading criteria. Table 3 (next page) shows the grades we gave the various policies discussed here.

Table 3. Grades for the various types of policies.

Policy	Power	No. of shutdowns	Shutdown accuracy	Interactive performance	Memory requirements	Computation requirements
Nonstationary requests (NS)	A	A	A	A	C	B
Time-index semi-Markov models (SM)	A	B	A	A	A	A
Device-dependent time-out (DDT)	A	B	A	A	A	A
Learning tree (LT)	B	B	B	B	C	C
Adaptive time-out policy (ATO)	B	A	A	B	B	A
Exponential average (EA)	B	C	C	C	B	B
Discrete-time Markov processes (DM)	C	A	C	A	A	A

Qualitative analysis

The exponential average predictive policy has many more shutdowns. Most of them actually waste energy because the disk sleeps shorter than T_{ms}. However, this policy is not designed for managing power states of hard disks. It is designed for X-server and telnet when no mechanic movements are involved and state transition overhead is low. This shows the importance of examining the assumptions and the limitations of a policy before applying it to a specific device.

The nonstationary stochastic policy has better power-saving ability than the stationary policy because NS can adjust for changing workloads; NS and SM also have high grades in interactivity. We do not find any policy with A grades in all columns. If a policy aggressively saves power (such as SM), it is likely to issue more shutdown commands and degrade performance. On the other hand, a policy can be conservative in power saving and issue fewer shutdown commands. While performance and accuracy improve, these policies consume more power. Finally, the resource requirements are also important. The nonstationary stochastic policy has excellent power savings but requires substantial amount of memory.

Improving prediction accuracy with high-level information

All policies surveyed in this article share these assumptions: Request generation can be modeled by a single requester and observing request arrival is an effective means for predicting the lengths of idle periods. A trend in power management research is to include higher-level information, particularly software-based information from compilers and operating systems.[18] One study classifies requests by processes and obtains encouraging results.[19] We think software will play an increasingly important role in power reduction.

THIS ARTICLE STUDIES system-level power management policies. We implemented and compared different policies on a laptop computer. Our experiments show that policy selection is a trade-off among power saving, performance, interactivity, and resource requirements. ■

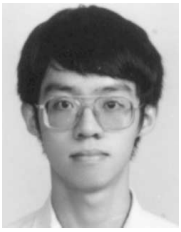
Acknowledgment

This work is supported by the Gigascale Silicon Research Corp. and the US National Science Foundation under grant CCR-9901190.

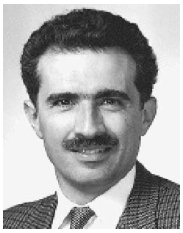
■ References

1. R. Golding et al., "Idleness is not Sloth," *Usenix Winter Conf.*, Usenix, San Diego, Calif., 1995, pp. 201-212.
2. L. Benini et al., "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. VLSI Systems*, vol. 8, no. 3, June 2000.
3. C.H. Hwang and A. C. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, 1997, pp. 28-32.
4. Standard Performance Evaluation Corp., http://www.spec.org.
5. F. Douglis et al., "Adaptive Disk Spin-Down Policies for Mobile Computers," *Computing Systems*, vol. 8, no. 4, 1995, pp. 381-413.

6. A. Karlin et al., "Competitive Randomized Algorithms for Non-Uniform Problems," *Algorithmica*, vol. 11, no. 6, June 1994, pp. 542-571.
7. D. Ramanathan and R. Gupta, "System Level On-Line Power Management Algorithms," *Proc. Design Automation and Test in Europe*, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 606-611.
8. J. Wilkes, Predictive Power Conservation, technical report HPL-CSP-92-5, Hewlett-Packard, Palo Alto, Calif., 1992.
9. M.B. Srivastava et al., "Predictive System Shutdown and Other Architecture Techniques for Energy Efficient Programmable Computation," *IEEE Trans. VLSI Systems*, vol. 4, no. 1, Mar. 1996, pp. 42-55.
10. E.-Y. Chung et al., "Dynamic Power Management Using Adaptive Learning Tree," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, 1999, pp. 274-279.
11. L. Benini et al., "Policy Optimization for Dynamic Power Management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 6, June 1999, pp. 813-833.
12. Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes," *Proc. Design Automation Conf.,* ACM Press, New York, 1999, , pp. 555-561.
13. T. Simunic, L. Benini, and G.D. Micheli, "Event-Driven Power Management of Portable Systems," *Proc. Int'l Symp. System Synthesis*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 18-23.
14. T. Simunic et al., "Dynamic Power Management for Portable Systems," *Int'l Conf. Mobile Computing and Networking*, ACM Press, New York, 2000, pp. 11-19.
15. E.-Y. Chung et al., "Dynamic Power Management for Non-Stationary Service Requests," *Proc. Design Automation and Test in Europe*, IEEE CS Press, 1999, pp. 77-81.
16. Y.-H. Lu et al., "Quantitative Comparison of Power Management Algorithms," *Proc. Design Automation and Test in Europe*, IEEE CS Press, 2000, pp. 20-26.
17. Advanced Configuration and Power Interface, http://www.teleport.com/~acpi/
18. Workshop on Compilers and Operating Systems for Low Power, http://www.cse.psu.edu/~kandemir/colp.html, Oct. 2000.
19. Y.-H. Lu et al., "Operating System Directed Power Reduction," *Proc. Int'l Symp. Low Power Electronics and Design*, ACM Press, New York, 2000, pp. 37-42.



Yung-Hsiang Lu is a PhD candidate in the electrical engineering department at Stanford University. His research interests include low-power design, especially system-level power management. Lu has an MS in electrical engineering from Stanford University. He is a student member of IEEE and ACM.



Giovanni De Micheli is professor of electrical engineering, and by courtesy, of computer science at Stanford University. His research interests include several aspects of design technologies for integrated circuits and systems, with particular emphasis on synthesis, system-level design, hardware-software codesign and low-power design. De Micheli has a Nuclear Engineer degree from the Politecnico di Milano and an MS and PhD in electrical engineering and computer science from the University of California at Berkeley. He is the author of *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, 1994) and coauthor and/or coeditor of four other books and over 200 technical articles. He is member of the technical advisory board of several EDA companies, including Magma Design Automation, Coware, and Aplus Design Technologies.

■ Direct questions or comments about this article to Yung-Hsiang Lu, Room 326 Gates Bldg., Computer Systems Laboratory, 353 Serra Mall, Stanford, CA 94305; luyung@stanford.edu.