January 2009

A homogeneous architecture for power policy integration in operating systems

Nathaniel Pettis

Yung-Hsiang Lu

Follow this and additional works at: <http://docs.lib.purdue.edu/ecepubs>

Pettis, Nathaniel and Lu, Yung-Hsiang, "A homogeneous architecture for power policy integration in operating systems" (2009).
Department of Electrical and Computer Engineering Faculty Publications. Paper 35.
<http://dx.doi.org/http://dx.doi.org/10.1109/TC.2008.180>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

A Homogeneous Architecture for Power Policy Integration in Operating Systems

Nathaniel (Eddie) Pettis, *Member, IEEE*, and Yung-Hsiang Lu, *Senior Member, IEEE*

Abstract—A significant volume of research has concentrated on operating system (OS)-directed power management. The primary focus of previous research has been the development of better policies. In this paper, we provide evidence that one policy may outperform another under different conditions. Hence, it is difficult, or even impossible, to design the “best” policy for all computers. We explain how to select the best policies at runtime without user or administrator intervention by using a software framework called the Homogeneous Architecture for Power Policy Integration (HAPPI). This architecture is portable across different platforms running Linux. HAPPI specifies common requirements for policies and provides an interface to simplify the implementation of policies in a commodity OS. Our approach allows these policies to be compared simultaneously to select the best policy among a set of distinct policies at runtime. Experimental results indicate that HAPPI achieves energy savings within 4 percent of the best individual policy for each device in several computing systems without a priori knowledge of workloads.

Index Terms—Dynamic power management, automatic policy selection, operating systems.

1 INTRODUCTION

OPERATING systems (OSs) manage resources, including processor time, memory space, and disk accesses. Due to the growing popularity of portable systems that require long battery life, energy has become a crucial resource for OSs to manage [1], [2], [3], [4]. Power management is also important in high-performance servers because performance improvements are limited by excessive heat [5], [6], [7]. Finding better policies has been the main focus of OS-directed power management research in recent years [8]. A policy is an algorithm that chooses when to change a component's power states and which power states to use.

Existing studies on power management assume that only one policy can be used to save energy and focus on finding the best policies for unique request patterns. Although some policies allow their parameters to be adjusted at runtime [9], [10], [11], the algorithms remain the same. Typically, the best policy for a hardware platform and application is determined by running an experiment or simulation multiple times with different policies. The underlying assumption of this method is knowledge of both the hardware and software of the target machine. The large number of existing policies suggests that different policies may be needed to achieve better energy savings in different scenarios. Since most studies evaluate their policies using a single hardware component, it is unclear whether the same policy can be applied to another similar component. For example, hard disks and CD-ROM drives are both block devices, but their workload behaviors are different. Furthermore, desktops,

laptops, and servers may utilize very different software packages with different behaviors.

In our previous work [12], we present a technique called automatic policy selection. Instead of choosing one policy in advance, a group of policies can be eligible at runtime, and one policy is selected in response to the changing request patterns. This is especially beneficial for a general-purpose system or on-demand computing where usage patterns can vary dramatically when the users execute different programs, as opposed to embedded systems where workload is relatively homogenous. Automatic policy selection removes the aforementioned assumptions from manual policy selection techniques by automatically adapting to different hardware platforms and the current workload on each device in the computing system. In this paper, we demonstrate that different policies improve energy savings for different devices, even for the same workload. Using this information, our objective is to utilize automatic policy selection to select the proper policy for a given application without user or administrator interaction.

This paper has the following five contributions:

- We demonstrate that different policies are necessary to improve energy savings for different devices and workloads by providing several practical examples.
- We demonstrate that automatic policy selection achieves comparable energy savings to the best single policy for each device in a computing system without knowing the proper policy for each device in advance.
- We apply automatic policy selection to server workloads and demonstrate that significant energy savings may be achieved.
- We verify the accuracy of Homogeneous Architecture for Power Policy Integration (HAPPI)'s estimation mechanisms using hardware measurement to validate HAPPI's assumption that policies may predict their own energy savings.

• N. Pettis is with Google, 1600 Amphitheatre Parkway, Mountain View, CA 94043. E-mail: epettis@google.com.

• Y.-H. Lu is with the Purdue University, 465 Northwestern Avenue, West Lafayette, IN 47907. E-mail: yunglu@purdue.edu.

Manuscript received 21 Nov. 2006; revised 12 Nov. 2007; accepted 3 Sept. 2008; published online 17 Sept. 2008.

Recommended for acceptance by N. Bagherzadeh.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0443-1106.

Digital Object Identifier no. 10.1109/TC.2008.180.

TABLE 1
Summary of Comparison Policies

Policy	Advantages	Disadvantages
2-COMP [13]	Saves energy for a variety of workloads	Does not excel with any workloads
ADAPT [9]	Works well with bursty workloads	May require long time to adapt
EXP [10]	Works very well for periodic accesses	Performs very poorly with bursty workloads
NSMARKOV [11]	Statistically optimal for stable access probabilities	Performs poorly for rapidly changing workloads

- e. We profile HAPPI's performance and reveal that our automatic policy selection incurs only 1.22 percent overhead beyond existing power management approaches that use a single policy, suggesting that automatic policy selection may be used in practice with little performance degradation.

The rest of this paper is organized as follows: Section 2 describes related work and introduces the four policies we compare in this study. Section 3 provides a series of experiments that support automatic policy selection as a useful technique to improve energy savings. Section 4 presents HAPPI, a software framework that enables automatic policy selection in Linux. Section 5 describes our experimental platforms and workloads. Section 6 illustrates how automatic policy selection chooses the best policy for each device in a variety of computing systems and workloads, improving energy savings beyond using a single policy on all devices. Section 7 validates HAPPI's ability to predict a policy's effectiveness with low-performance overhead. Section 8 discusses the limitations of automatic policy selection and opportunities for future study. We provide concluding remarks in Section 9.

2 RELATED WORK

2.1 Dynamic Power Management

Most users are familiar with power management for block access devices, such as hard disks. Users can set the timeout values in Windows' Control Panel or using Linux's `hdparm` command. This is the most widely used "timeout policy." Hundreds of power management policies have been proposed covering each level of the computing hierarchy [8]. A search in IEEE Xplore yields 104 results for "power management" and "policy." The same search returns 875 results from the ACM Digital Library. In this work, we focus on OS-level policies. Several OS-level policies use adaptive parameters to adjust to changes in workloads [9], [10], [11]. Other policies include stochastic optimization to improve energy savings [11], [14], [15], [16], [17]. Although these policies adapt to changes in hardware and workloads, our method actually changes policies at runtime in response to changes in workloads.

Several other policies focus on dynamic voltage scaling [18], [19], [20], [21] and memory management [22], [23], [24]. We currently consider only policies for I/O devices. Specifically, we use four existing policies: two-competitive timeout [13], adaptive timeout [9], exponential averages [10], and nonstationary Markov models [11]. Although we use only four policies, the techniques described in this work may be applied to other policies as well by utilizing the interface developed in Section 4. Table 1 summarizes the

advantages and disadvantages of each policy. This table is provided for reference when comparing policies in later sections.

2.1.1 Two-Competitive Timeout Policy [13]

It uses a fixed timeout length equal to the break-even time of the device. The *break-even time* is defined as the amount of time a device must be shut down to save energy and may be computed as $t_{be} = \frac{E_{wake}}{P_{on} - P_{off}}$, where E_{wake} is the awakening energy, P_{on} is the power consumption in the active state, and P_{off} is the power in the sleep state. We abbreviate the two-competitive timeout policy as "2-COMP." This policy consumes no more than twice the energy of an optimal policy with full knowledge of future accesses. 2-COMP is considered a conservative policy because it waits until significant idleness has occurred before changing states.

2.1.2 Adaptive Timeout Policy [9]

It changes the timeout value to improve energy savings as workloads change. We abbreviate the policy as "ADAPT." In this paper, we use the arithmetic version of ADAPT. This version increases or decreases its timeout by a fixed amount after each access. We use an initial timeout prediction of t_{be} and arithmetic increment of $0.1t_{be}$. ADAPT is also considered a conservative policy because it waits until idleness has occurred before changing states. However, ADAPT is more aggressive than 2-COMP because ADAPT may shorten its timeout length to save additional energy.

2.1.3 Exponential Average Policy [10]

It predicts a device's idleness and makes decisions to shut down a device immediately following each access. We abbreviate the exponential average policy as "EXP." This policy uses the recursive relationship $I[n+1] = \alpha i_n + (1 - \alpha)I[n]$ to predict the idleness after the current access $I[n+1]$ from the previous prediction $I[n]$ and the previous actual idle length i_n . The parameter α is a tunable parameter ($0 \leq \alpha \leq 1$) that determines how much to weight the most recent idle length. The authors in [10] suggest $\alpha = 0.5$. EXP is an aggressive policy because it changes states before further idleness has occurred.

2.1.4 Nonstationary Markov Policy [11]

This models device accesses using Markov chains. We will abbreviate this policy as "NSMARKOV." At fixed periods, called time slices, NSMARKOV computes a transition probability matrix for the device. This matrix contains the probability of a request given whether an access occurred during the previous time slice. At each time slice, NSMARKOV uses the matrix's measurement to index into a lookup table that specifies the probability of issuing each power transition command. NSMARKOV is the most

aggressive of our sample policies because it changes states before further idleness has occurred and may change states even after a long burst of accesses.

2.2 Operating System-Directed Power Management

The Advanced Configuration and Power Interface (ACPI) specification [25] defines a platform-independent interface for power management. ACPI describes the power consumption of devices and provides a mechanism to change the power states. However, ACPI requires an OS-directed power manager to implement policies. Microsoft Vista [26] uses ACPI to allow individual devices' power states to be controlled by the device driver, which presumably implements a single policy such as those mentioned in Section 2.1. Microsoft Vista provides a command line interface to set device timeout values, processor throttling thresholds, and system shutdown timeouts. Linux handles power management similarly to Vista using ACPI [27] and user-space applications with administrative privilege, such as `hdparm`, to modify timeouts and policies. In contrast to Windows and Linux, HAPPI's policies manage power states above the driver level because our method allows complex policies to operate on multiple devices simultaneously, enhancing code reuse. Furthermore, HAPPI differs from existing OS power management mechanisms by automatically selecting the best policy at runtime, whereas Windows and Linux require users or administrators to select and configure a single policy manually.

3 THE NEED FOR AUTOMATIC POLICY SELECTION

Section 1 describes how existing studies on power management perform repeated experiments and rely upon the assumption that both the hardware and software are known. This section demonstrates that request patterns may vary significantly among workloads. This section also demonstrates that different devices may experience different request patterns for the same workload. This evidence implies that different policies may be necessary to achieve better energy savings. For general-purpose machines, many different combinations of software applications may be executed at any given time. Since it is unlikely that experiments will be performed to cover all possible workload mixes, the best power management policy should be selected automatically, based upon both the hardware and the workload.

Fig. 1 depicts the access probability for a web server running SPECWeb99. The server contains three identical disks. Disk 1 is the system disk and manages the access log for the server. Disk 2 and Disk 3 each contain half of the server's data. Disk 2 contains the lower half, sorted alphabetically. Disk 3 contains the upper half. The access probability is computed using the state machine [11] shown in Fig. 1a. Every 5 seconds, the state machine determines if an access has occurred, and the state machine should transition to the REQ state. Otherwise, the state machine transitions to the NREQ state. Each plot in Fig. 1b indicates the probability that a request is followed by another request within 5 seconds (REQ/REQ) and the probability that an idle interval is followed by another 5-second idle interval (NREQ/NREQ). The vertical axis represents the probability. A high probability for REQ/REQ indicates a bursty workload. In

Fig. 1. (a) State machine for computing access probability. (b) Access probabilities for three disks in a web server as the number of simultaneous connections increases from 20 to 460.

contrast, a low probability represents intermittent accesses. A high probability for NREQ/NREQ indicates that idle intervals are very long, whereas a low probability suggests that idle intervals are brief. When REQ/REQ is low and NREQ/NREQ is high, many opportunities exist to save energy. The horizontal axis represents time. In this workload, we increase the number of connections for SPECWeb99 from 20 to 460 in intervals of 40 connections over a 16-hour span. The separation between connection counts is noted by a solid vertical line.

We observe two significant trends from this figure. First, Disk 2 has significantly more idleness than Disk 1 and Disk 3 throughout the workload. Disk 2 has low REQ/REQ and high NREQ/NREQ. Since Disk 2's data are accessed frequently, most of its data are located in the file cache. In contrast, the data on Disk 3 are less popular and are more likely to be evicted from the file cache, leading to additional disk accesses. Other studies support the observation that fewer disk accesses are needed for popular data [28], [29]. Disk 1 contains the access log, which handles many file writes. These writes must be committed to the disk frequently by the dirty page writeback thread, leading to several disk accesses. The second trend is that the REQ/REQ probability changes dramatically within a connection range. This is due to compulsory misses in file cache as new files are accessed. Policies that adapt to changes in workload must adapt quickly to discard the bursty history to effectively save energy within a workload.

Table 2 shows the energy savings achieved by running the four policies in Section 2.1 on the SPECWeb99 workload. These results support the previous claims. 2-COMP and ADAPT do not waste energy for Disk 1 and Disk 3 during bursty workloads because they are conservative policies. In contrast, the aggressive EXP and NSMARKOV

TABLE 2
Energy Savings for Four Policies
Running SPECWeb99 Workload

Device	2-COMP	ADAPT	EXP	NSMARKOV
Disk 1	2.8%	10.5%	-4.1%	-14.1%
Disk 2	28.1%	-15.5%	48.6%	53.1%
Disk 3	9.0%	15.5%	-3.6%	1.8%

Best policy for each device is boldfaced.

TABLE 3
Energy Savings for Four Policies on Two Desktops Running
Interactive Workload Described in Section 5.2.1

Device	2-COMP	ADAPT	EXP	NSMARKOV
Disk 1	4.5%	16.6%	4.4%	22.8%
CD-ROM 2	12.2%	14.4%	14.7%	17.0%
Disk 2	37.1%	46.1%	29.0%	37.6%
CD-ROM 2	12.5%	16.1%	10.2%	-5.4%
Network 2	3.6%	3.6%	-9.4%	-24.5%

Best policy for each device is boldfaced.

policies frequently mispredict these transitions. ADAPT saves more energy than 2-COMP because ADAPT shortens its timeout when the disk becomes idle. NSMARKOV outperforms all policies for Disk 2 because the device has stable statistical properties. Note that 2-COMP saves energy for all devices, but a significant energy gap exists between 2-COMP and the best policy for each device. This evidence supports the common use of fixed timeout policies because the policy works “well enough” for a variety of workloads. In fact, Table 2 indicates that ADAPT, EXP, and NSMARKOV each waste energy on at least one disk. Hence, using any of these policies consume more energy on some devices than using no power management. This experiment shows that no single policy can save the most energy for different disks, even when running the same workload.

Table 3 provides the energy savings for the four policies on two different desktops running the same interactive workload. We use two different desktops to demonstrate that different power parameters imply that different policies should be used, even for the same workload. These desktops have different hardware configurations. We observe that NSMARKOV achieves the highest energy savings for both devices in Desktop 1. In contrast, ADAPT achieves the highest energy savings for the disk and CD-ROM in Desktop 2. Both 2-COMP and ADAPT achieve the same energy savings for the network card.

These experiments illustrate an important observation regarding policy selection. These results suggest that the best policy depends on both hardware and workload. Since the best policy varies between devices and workload, the best policy should be determined at runtime based upon the current workload. Moreover, the policy should be selected based on the device parameters. Hence, it is important to perform automatic policy selection. The next section describes our method to determine the best policy.

4 AUTOMATIC POLICY SELECTION USING HAPPI

In this section, we present a software framework that allows automatic policy selection to be implemented within the Linux kernel. This design specifies homogeneous requirements for all policies so they can be integrated into the OS

Fig. 2. Organization of HAPPI within the Linux kernel.

and selected at runtime. Homogeneous requirements are necessary to allow significantly different policies to be compared by the OS. We refer to this architecture as HAPPI. HAPPI is currently capable of supporting power policies for disk, CD-ROM, and network devices but can easily be extended to support other I/O devices. To implement a policy in HAPPI, the policy designer must provide

1. a function that predicts idleness and controls a device's power state and
2. an estimator function that accepts a trace of device accesses, determines the actions the control function would take, and returns the energy consumption and access delay from the actions.

Fig. 2 depicts the organization of HAPPI within the Linux kernel. User-space applications issue device requests through file descriptors and sockets. Both of these request types are serviced by the virtual file system (VFS). HAPPI records each of these accesses, forwards the accesses to the device driver, and issues a notification to the active policy. The active policy is selected by the evaluator using the estimator functions for all policies. The active policy has the exclusive permission to determine the power states of the device. A policy may update its predictions and request device state changes on each device access or a periodic timer interrupt. For a more detailed explanation of HAPPI's implementation, we direct the reader to our technical report [30].

4.1 Policy Set

Each device has a set of policies that are capable of managing the device. A policy is said to be *eligible* to manage a device if the policy is in the device's policy set. A policy becomes eligible when it is loaded into the OS as a kernel module and is no longer eligible when it is removed from the OS. A policy may be inserted or removed at runtime. The policy is *active* if it is selected to manage the power states of a specific device by HAPPI. Each device is assigned only one active policy at any time. However, a policy may be active on multiple devices at the same time by creating data structures for each device within the policy and multiplexing HAPPI function calls. When a policy is activated, it obtains exclusive control of the device's power state. The policy is responsible for predicting idleness, determining when the device should be shut down, and requesting state changes. An active policy may update its predictions and request device state changes on each device access or after a specified timeout.

4.2 Measuring Device Accesses

Policies monitor device accesses to predict idleness and determine when to change power states. We refer to the data required by policies to make decisions as *measurements*. One such measurement is a trace of recent accesses. Policies use access traces to make idleness predictions. Whenever the device is accessed, HAPPI captures the size and the time of the access. More advanced policies may require additional measurements, such as a probability distribution of accesses. HAPPI also records the energy and the delay for each device. Energy is accumulated periodically and after each state transition. We define delay as the amount of time that an access waits for a device to awaken. We only accumulate delay for a process's first access while the device is sleeping or awakening because Linux prefetches adjacent blocks on each access. Delay may be used to determine power management's impact on system performance.

4.3 Policy Selection

Policy selection is performed by the *evaluator*. The evaluator periodically asks all eligible policies to provide an estimate for the current measurements. An *estimate* consists of energy consumption and total delay for the measurement data and quantifies a policy's ability to manage the device. To accomplish this, each policy must provide an estimation function that uses HAPPI's measurement data to analyze what decisions the policy would have made if it were active when the measurements were taken. The energy and the delay for these decisions are computed by the estimation function and returned to the evaluator. We study the accuracy of estimates in Section 7.1.

To illustrate how an estimator works, we provide an example. Fig. 3 shows an estimator function for a fixed-timeout policy, such as 2-COMP. The estimator iterates over all device accesses, determining the time between each access. If the idleness t_{idle} exceeds the timeout length t_o (lines 8-12), the device shuts down and consumes $P_{on}t_o$ energy in the idle state. The remaining $(t_{idle} - t_o)$ time is spent in the off state. The device consumes E_{wake} energy to awaken from the off state before servicing the request. Since the device must change power states, a delay of t_{wake} is incurred before the access begins. However, if $t_{idle} < t_o$ (lines 13-14), the device does not shutdown and consumes energy $P_{on}t_{idle}$ before serving the next request. No delay occurs before the request begins. After all accesses have been considered, the total energy consumption and delay is returned to HAPPI.

An active policy for each device is selected by the evaluator after it receives estimates from all policies. The evaluator selects each active policy by choosing the best estimate for an evaluation metric, such as total energy consumption or energy-delay product. If another policy's estimate is better than the currently active policy, the inferior policy is deactivated and returned to the set of eligible policies. The superior policy is activated and assumes control of the device's energy management. Otherwise, the current policy remains active. The policy set always includes a *null policy* (abbreviated as "NULL") that keeps the device in the highest power state to achieve the best performance. If the null policy produces the best estimate, none of the eligible power management policies can save power for the current workload. Under this condition, power management is disabled

```

FIXED-TIMEOUT-ESTIMATOR(accesses)
    /*  $t_o$  = timeout length */
    /*  $P_{on}, P_{off}, P_{active}$  = power states */
    1  energy  $\leftarrow$  0
    2  delay  $\leftarrow$  0
    3  last  $\leftarrow$  accesses[1].time
    4  for each  $q \in$  accesses[2 : n]
    5      do now  $\leftarrow$  q.time
    6       $t_{idle} \leftarrow$  now - last
    7      if  $t_{idle} \geq t_o$ 
    8          then /* Device will change power states */
    9              energy  $\leftarrow$  energy +  $P_{on}t_o$ 
   10              energy  $\leftarrow$  energy +  $P_{off}(t_{idle} - t_o)$ 
   11              energy  $\leftarrow$  energy +  $E_{wake}$ 
   12              delay  $\leftarrow$  delay +  $t_{wake}$ 
   13          else /* Device will not change power states */
   14              energy  $\leftarrow$  energy +  $P_{on}t_{idle}$ 
   15      last  $\leftarrow$  now
   16  return (energy, delay)
    
```

Fig. 3. Sample estimator function for a fixed-timeout policy.

until the next evaluation. The evaluator determines when reevaluation should occur. In this paper, we evaluate policies every 20 seconds and select the policy with the lowest average power. We direct the reader to our technical report for the experiments used to derive this 20-second interval [30].

5 EXPERIMENTAL SETUP

This section describes our experimental setup and benchmarks. We use physical measurements to demonstrate that our approach can save comparable energy to the best individual policy without a priori knowledge of hardware or workload. We apply automatic policy selection to desktops and servers to show that HAPPI can improve energy savings in a wide range of systems.

5.1 Power Measurements

We use two different hardware configurations in our experiments: a 1.8-GHz IBM desktop computer with 512 Mbytes of RAM and a 4-CPU, 500-MHz Dell PowerEdge server with 1 Gbyte of RAM. We manage three devices for our desktop: a Seagate 3.5" disk, a Mitsumi CD-ROM drive, and a Linksys NC100 PCI network card. Our server contains three identical IBM UltraStar SCSI disks. We use a wide assortment of devices to demonstrate that automatic policy selection is necessary to save energy for different devices. We determine the power parameters for the desktop devices by experimental measurement using a National Instruments data acquisition card (NI-DAQ). Fig. 4a shows our experimental setup. Fig. 4b shows a close-up view of the PCI extender card to measure energy consumption for the network card. Table 4 lists the information required by the ACPI specification for each device. The active state is the state where the device can serve requests. The sleep state is a reduced power state in which requests cannot be served. Changing between states incurs energy and wakeup delay shown in Table 4. For reference, we include the break-even time of each device. All energy savings for the desktop are based upon physical measurement using the NI-DAQ card. To determine the server's energy savings, we use the power model from [31] and HAPPI's software energy measurements because

Fig. 4. Photograph of experimental setup. (a) Complete system with sensing resistors and NI-DAQ junction box. (b) Using PCI extender to measure the power of the Linksys network card.

HAPPI does not yet support state transitions for SCSI devices. The power parameters for the IBM UltraStar disks are shown in Table 4. We emulate disk shutdowns by delaying requests in each block queue during state transitions, as proposed in [32].

We instrument device accesses and policy selection by printing information to Linux's kernel log. The kernel log is a circular buffer in kernel memory. Hence, data collection does not introduce any additional device accesses. When the experiment is complete, the results and the kernel log are written to the system disk for analysis. We use a 1-Mbyte kernel log, allowing approximately 3 hours of data. For longer benchmarks, we record the kernel log after each hour. We use the 2.6 kernel's `laptop_mode` option to facilitate power management. This option reduces the number of disk accesses by delaying writes to disk until the disk services a read request or a large number of writes have been buffered. Without `laptop_mode`, the disk is accessed at least once every 5 seconds and is never idle long enough to save energy. We also adjust the commit interval of journaling file systems, such as `ext3` and `ReiserFS`, because `laptop_mode` does not delay commits.

5.2 Benchmarks

To illustrate HAPPI's ability to track changes in workloads and select policies, we execute applications that provide a range of activities for HAPPI to manage. We consider two workloads: interactive and web service.

TABLE 4
Power States for Devices

		Disk		CD-ROM	Network
		Seagate	UltraStar	Mitsumi	Linksys
Active	Power	5.17 W	6.0 W	2.16 W	0.095 W
Sleep	Power	1.6 W	1.4 W	1.67 W	0.063 W
	Wakeup Delay	5.0 s	10.1 s	7.0 s	4.0 s
	Wakeup Energy	19.0 J	107.9 J	16.1 J	0.325 J
	Break-even Time	5.3 s	23.5 s	32.9 s	10.2 s

Network card delay includes time to reestablish network connectivity in software.

5.2.1 Interactive

This benchmark models a user executing interactive programs on a desktop computer. We consider five individual workloads similar to those performed by a desktop user:

- Workload 1: Web browsing and buffered media playback from CD-ROM.
- Workload 2: Download video and buffered media playback from disk.
- Workload 3: CVS checkout from remote repository.
- Workload 4: E-mail synchronization and sequential access from CD-ROM.
- Workload 5: Kernel compile.

5.2.2 Web Service

This workload executes the SPECWeb99 benchmark with increasing numbers of connections on an Apache 2.2 Web server. The workload begins at 20 connections and increases to 460 connections in increments of 40 connections after every 80 minutes over a span of 16 hours. Server logs and system files are stored on Disk 1. Data files are stored on Disk 2 and Disk 3. Disk 2 contains the lowest half of the files, sorted in alphabetical order. Disk 3 contains the upper half.

6 IMPROVING ENERGY SAVINGS AT RUNTIME

This section applies HAPPI to different computing systems with the workloads in Section 5.2 and demonstrates that automatic policy selection achieves comparable energy savings to the best individual policy. We further demonstrate that different policies should be selected for each device.

6.1 Desktop Applications

Fig. 5a illustrates the policies' estimates for each workload described in Section 5.2.1. On this figure, the horizontal axis indicates time. The vertical axis represents each estimate's average power. The Gantt chart in Fig. 5b summarizes the estimates by indicating the selected policy at each evaluation interval. A cross ("x") indicates the selection of the policy on the vertical axis at the time indicated on the horizontal axis. Vertical bars separate workloads.

Fig. 5. Policy selection for interactive workload. (a) Estimated energy consumption for each policy on two devices. (b) Selected policies for devices at each evaluation.

We begin by observing the estimates for the disk. Fig. 5a reveals that NSMARKOV begins saving energy very quickly after the experiment begins. A dirty page writeback takes place at point A. EXP is selected because it accurately predicts the following idleness and aggressively shuts down the device to save energy. At points B and C, EXP shuts down the disk before another burst arrives. EXP's estimate rises sharply to indicate mispredictions. The 2-COMP, ADAPT, and NSMARKOV estimates do not increase as sharply because these policies do not shut down the device before the next burst. Since multiple bursts exist, ADAPT is selected to manage power states more conservatively until the bursts complete at the start of Workload 2. During span D in Workload 4, accesses are predictably spaced, suggesting that EXP would be a favorable policy. However, EXP requires time to refine its prediction. NSMARKOV controls the disk until EXP can accurately predict the idleness. In contrast, 2-COMP and ADAPT require longer to shut down when idleness exists, so these policies save less energy. At point E, some idleness exists but interarrivals are erratic. Since NSMARKOV uses statistical information rather than the interarrival time of consecutive accesses, NSMARKOV predicts the idleness well and saves energy.

The CD-ROM exhibits a more bursty workload than the disk. The beginning of Workload 1 exhibits this behavior and is indicated at point F by a high EXP estimate and a low NSMARKOV estimate. At point G, several accesses occur to read a new audio track. NSMARKOV mispredicts on these accesses due to the prior idleness, and the selected policy briefly changes to ADAPT. After the burst completes, NSMARKOV is selected again. We observe at points H and J that EXP mispredicts bursts and is unable to save as much energy as the other policies. The CD-ROM is idle during span K. EXP's estimate increases because the policy mispredicts the last access and does not shut down the CD-ROM during span K. All other estimates improve because they predict the last access correctly. However, the energy estimates are different in magnitude because the policies shut down the CD-ROM after different amounts of time. During Workload 4, more bursty accesses occur.

NSMARKOV is selected to exploit idleness immediately following the bursts.

Table 5 verifies that HAPPI achieves energy savings similar to the best individual policy. HAPPI consumes 3.1 percent more energy than NSMARKOV for the disk and 0.5 percent *less* energy for the CD-ROM. Hence, HAPPI is capable of choosing the best policy for each device from a set of policies for an unknown workload at runtime.

This experiment compares several distinct policies simultaneously on different devices and provides insight into policies' properties that make them effective for different workloads and devices. The disk experiences a mix of bursty accesses and isolated accesses, but NSMARKOV remains an effective policy. The disk has a short break-even time (5.3 seconds), meaning that shutting down the device early does not incur a significant energy penalty. Furthermore, most idle periods exceed the disk's break-even time, providing many opportunities to shut down the device and save energy. Hence, exploiting idleness quickly is more important for this disk than predicting correctly. The CD-ROM has a much longer break-even time (32.9 seconds) than the disk and predominantly bursty accesses. 2-COMP and ADAPT require very long idleness to expire before shutting down, and EXP predicts bursts poorly. However, the CD-ROM has statistically stable access patterns. NSMARKOV predicts idleness more effectively than the other policies.

TABLE 5
Energy Consumption and Savings for All Individual Policies and HAPPI for Interactive Workload on Desktop Computer

Policy	Disk		CD-ROM	
NULL	26,599 J	—	11,405 J	—
2-COMP	25,397 J	4.5%	10,016 J	12.2%
ADAPT	22,173 J	16.6%	9,762 J	14.4%
EXP	25,433 J	4.4%	9,726 J	14.7%
NSMARKOV	20,540 J	22.8%	9,471 J	17.0%
HAPPI	21,353 J	19.7%	9,411 J	17.5%
Difference	813 J	3.1%	-60 J	-0.5%

Data are obtained from physical measurement.

Fig. 6. Policy selection for server workload. (a) Estimated energy consumption for each policy on devices. (b) Selected policies for devices at each evaluation.

6.2 Server Applications

Fig. 6a illustrates the estimates for each policy on each of the three disks in our server running the benchmark in Section 5.2.2. Vertical bars separate increasing numbers of connections. Fig. 6b indicates the selected policies. At points A and B in Fig. 6a, we see a cyclical change in estimates for Disk 1. The rise and fall of estimates closely track the access probability from Section 3. We further note that NSMARKOV's estimate exceeds NULL at A and B. Fig. 6b indicates that 2-COMP is selected at point A and ADAPT is selected at point B. Here, HAPPI improves energy savings by avoiding a policy that makes poor decisions. We note that HAPPI selects a conservative policy at the beginning of each workload to reduce the number of mispredicted shutdowns. At point C, accesses occur frequently enough that NSMARKOV consistently mispredicts. HAPPI selects ADAPT because it saves energy and predicts bursts of activity correctly. During span D, no policies can save sufficient energy and little variation exists between estimates.

Disk 2 has a very different request pattern from Disk 1. Significant idleness exists throughout most of the workload, providing aggressive policies such as NSMARKOV with many opportunities to save energy. 2-COMP is too conservative to save a significant amount of energy. At point E, we note that 2-COMP has a much higher estimate than the other policies because ADAPT, EXP, and NSMARKOV change their behavior to exploit the increased idleness. During span F, 2-COMP is rarely able to save energy

between accesses, unlike the other policies. For Disk 2, HAPPI saves energy by choosing the policy that most aggressively exploits idleness.

Disk 3 exhibits similar behavior to Disk 1. During the first part of the workload (indicated by point G), Disk 3 is idle. Hence, all policies' estimates decrease quickly. After the initial idleness, Disk 3 exhibits a cyclic access probability similar to Disk 1. HAPPI's policy selections are also similar, choosing ADAPT at the beginning of each workload and increasing the aggressiveness of the policy as accesses become more interspersed. During span H, insufficient idleness exists to save significant energy, so estimates have little difference.

Table 6 indicates the energy consumption for each disk in our server for each policy. ADAPT is the best policy for Disk 1 and Disk 3. ADAPT adjusts its timeout value to save more energy than 2-COMP during periods of idleness. Furthermore, ADAPT predicts bursts more effectively than EXP and NSMARKOV because it must wait until the timeout expires before shutting down the device. NSMARKOV is the best policy for Disk 2 because its access probability is stable across the entire workload, allowing NSMARKOV to aggressively shut down the disk. In contrast, 2-COMP and ADAPT must wait until their timeouts expire. NSMARKOV saves more energy than EXP because the interarrival time between accesses is not always predictable. ADAPT controls Disk 2 poorly because the policy significantly mispredicts idleness after the number of connections exceeds 420. HAPPI achieves energy savings within 4.0 percent, 0.8 percent, and 2.1 percent

TABLE 6
Energy Savings for Several Policies Running SPECWeb99 Workload

Policy	Disk 1		Disk 2		Disk 3	
NULL	345.6 kJ	—	345.6 kJ	—	345.6 kJ	—
2-COMP	335.8 kJ	2.8%	248.6 kJ	28.1%	314.5 kJ	9.0%
ADAPT	309.4 kJ	10.5%	399.1 kJ	-15.5%	292.1 kJ	15.5%
EXP	359.6 kJ	-4.1%	177.5 kJ	48.6%	333.3 kJ	3.6%
NSMARKOV	394.5 kJ	-14.1%	162.2 kJ	53.1%	339.4 kJ	1.8%
HAPPI	323.2 kJ	6.5%	164.8 kJ	52.3%	299.2 kJ	13.4%
Difference	13.8 kJ	4.0%	2.6 kJ	0.8%	7.1 kJ	2.1%

Best policy for each device is boldfaced.

TABLE 7
Estimator Accuracy Each Policy on Experimental Hardware

Policy	Seagate	Mitsumi	Linksys
2-COMP	-10.4%	-14.3%	14.2%
ADAPT	4.1%	-13.9%	5.0%
EXP	-11.5%	-3.4%	8.2%
NSMARKOV	-13.2%	-7.7%	18.4%
Relative Error	17.3%	10.9%	13.4%

Maximum absolute error for each device shown in boldface.

of the best policies for of Disk 1, Disk 2, and Disk 3, respectively.

Recall from Section 6.1 that the disk experienced bursty accesses, but NSMARKOV was selected most frequently. The server's disks have a much larger energy penalty and break-even time than the desktop disk. Hence, policies must be more conservative during bursty behavior for server disks than for desktop disks. The policy selection is not solely dependent on hardware parameters. The desktop CD-ROM has a longer break-even time (32.9 seconds) than the server disks (23.5 seconds). However, the access patterns for CD-ROMs and disks are different. CD-ROMs usually experience sequential bursts of accesses with long periods of idleness between bursts. In contrast, disks may experience frequent bursts with both long and short periods of idleness between bursts. Hence, when workloads are unknown a priori, automatic policy selection is necessary to achieve energy savings.

7 EVALUATING ACCURACY AND OVERHEAD

7.1 Accuracy of Estimator Models

Accurate power models are required by estimates to determine the correct power policies for each device. This section performs a series of physical measurements to determine the accuracy of estimators compared to the hardware they model. We do not evaluate the UltraStar's accuracy because we use a software model rather than physical measurement. We run each workload on the hardware and compare the estimates at each time interval to the hardware measurements. We define the absolute error as the average difference between the estimator's energy prediction and the physical measurement. Table 7 displays the absolute error for each policy. We define the relative error as the difference between the most accurate and the least accurate estimator for each device. The relative error between estimators is more important than the absolute error because HAPPI needs only to choose the best policy from the eligible policies. We observe relative errors of 17 percent, 11 percent, and 13 percent between policies for the Seagate, Mitsumi, and Linksys, respectively. The accuracy of these estimators is dependent upon the power model. The power model should be available from ACPI, but few I/O devices support the standard. To overcome this obstacle, we use state-based power models in our estimators and determine power consumption for our devices through physical measurement. Many papers [33], [34], [35], [36], [37], [38] have studied power models. In HAPPI, power models may also be inserted as kernel modules. This mechanism allows power models to be improved independently of policies.

TABLE 8
Profiling Results Using Policies in Section 2.1

	Samples	Execution Time
Recording	47,832	0.16%
Evaluation	70,503	0.24%
Selection	64,850	0.22%
Policy	2,223	0.01%
Other	179,051	0.60%
Total	364,459	1.22%

Total number of samples is 29,967,835. Each sample equals one real-time clock tick.

7.2 Performance Overhead

We use oprofile [39] to quantify the computational overhead for automatic policy selection. HAPPI consists of three types of overhead beyond the traditional single-policy power management approach: recording access history, policy estimation, and policy selection. To compute the performance overhead from HAPPI, we run the benchmarks described in Section 5.2.1 and add the execution times of the history, estimation, and selection functions. A summary of profiling results for the configuration we use in our experiments is shown in Table 8. "Recording" includes all function calls to store accesses in a circular buffer. "Evaluation" consists of calls necessary to request, compute, and compare policies' estimates. "Selection" considers all functions necessary to change between two policies. "Policy" contains the decisions necessary to change power states. "Other" contains all function calls shared by multiple components of HAPPI. This configuration uses five policies, including those in Section 2.1 and NULL. Profiling indicates that 1.22 percent of all execution time is HAPPI overhead. Of this overhead, 0.16 percent is spent recording access history, 0.24 percent of execution time is spent evaluating policies, and 0.22 percent is spent selecting policies. An additional 0.60 percent is spent executing other HAPPI code shared between all HAPPI components. The individual policies execute with less than 0.01 percent performance overhead. The cumulative execution time of all HAPPI components is 1.22 percent. Hence, automatic policy selection causes little decrease in system performance, implying that it is practical for a variety of systems, including high-performance computers.

Fig. 7 depicts HAPPI's performance as the number of policies varies from 5 to 25. The percentage of execution time is decomposed into the five components from Table 8. To increase the number of policies, we create copies of each policy and insert the copied policies as separate kernel modules. One copy of each policy is added for each data point. We observe a slight increase in overhead as the number of policies increase due to longer evaluation. The time required for other components remains nearly constant across the different numbers of policies. When 25 policies are eligible, the total overhead is less than 1.8 percent. These results indicate that HAPPI is capable of supporting many policies with acceptable overhead. The overhead to record access history is independent of the number of policies. The evaluation function overhead is proportional to the number of policies in the system and their complexity. Since evaluation occurs infrequently (every 20 seconds), estimation's impact on performance is small.

Fig. 7. Performance overhead for various numbers of policies.

8 DISCUSSION

Our experiments in Section 6 indicate that HAPPI achieves energy savings within 4.0 percent of the best policy and avoids selecting policies that waste energy. However, additional energy savings beyond the best individual policy are difficult. Policy selections inherently lag changes in workloads because estimates are constructed from a buffer of past accesses.

HAPPI is both a framework to implement existing policies and a research tool to develop new innovative policies. Many existing studies use trace-based simulations to compare policies' abilities to save energy. HAPPI provides an environment to compare policies simultaneously in a real machine with real applications, allowing a more accurate comparison of policies. Thus, repeatable workloads are unnecessary to determine which policy outperforms the others. Furthermore, HAPPI's implementation allows modular policies that may be applied to many different devices and shared within the research community for better comparison.

We implement HAPPI in Linux, but we speculate that automatic policy selection may be implemented within the Windows Driver Foundation [40] in one of two ways. The first method involves implementing multiple policies within each device driver. The second method implements HAPPI's mechanisms within the kernel and policies software-only filter drivers. It is unknown how feasible this mechanism is in practice, since Windows' source code is proprietary.

To perform power management in Linux, we enable `laptop_mode` and increase the commit interval of journaling file systems. Increasing the commit interval increases the amount of data loss during power failure. This risk of data loss is a fundamental property of dirty page writeback and unavoidable even if we disable `laptop_mode`. The aforementioned environment setup merely increases the window of vulnerability. For our experiments, we use a 5-minute commit interval.

In Section 6.1, we do not perform power management on the network card because the workloads' simultaneous dependencies on the network card and the block devices make prediction difficult. The network card has a much lower power consumption than the disk and the CD-ROM. Hence, a process that accesses the disk or the CD-ROM and the network card in sequence may waste a significant amount of energy during the 4 seconds required for the

network card to awaken. For example, the disk consumes 20.7 J waiting for the network card to awaken. The network card must sleep for 217.7 seconds to save more energy than the disk consumes during the network card's wakeup delay. Since this rarely occurs in our workloads and HAPPI currently considers each device's policy independently, we disable power management of the network card. We note that HAPPI allows a single policy to control multiple devices simultaneously, permitting policies to be designed for reducing the total energy consumption of multiple devices. This represents an area for future research in automatic policy selection.

We implement HAPPI within the Linux kernel directly above the device driver-level because 1) it allows policies to be written in a device-independent manner and 2) it does not require modifying any of the device drivers that constitute 70 percent of the Linux source code [41]. At this level, a policy may be used to control any type of I/O device, including hard disks, CD-ROMs, and network cards, without modification.

9 CONCLUSION

We have proposed a software framework called HAPPI to simplify the implementation, configuration, and automatic selection of policies. We validate HAPPI's energy savings by implementing the architecture in Linux. Our experiments indicate that the best policy is dependent on a device's power parameters and workload. HAPPI simplifies this configuration process by automatically selecting the proper policy for each device, achieving energy consumption within 4 percent of the best individual policy without a priori knowledge of the workloads.

ACKNOWLEDGMENTS

Nathaniel Pettis is sponsored by the US Department of Education GAANN Fellowship and the Bilsland Dissertation Fellowship. The authors would like to thank David Canada and Karl Herb for their assistance and the anonymous reviewers for their valuable suggestions. This work was supported in part by US National Science Foundation NSF CAREER CNS-0347466 and by the Purdue Research Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

- [1] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, and R.P. Doyle, "Managing Energy and Server Resources in Hosting Centers," *Proc. ACM Symp. Operating Systems Principles (SOSP '01)*, pp. 103-116, 2001.
- [2] C.S. Ellis, "The Case for Higher-Level Power Management," *Proc. Workshop Hot Topics in Operating Systems (HotOS '99)*, pp. 162-167, 1999.
- [3] R. Neugebauer and D. McAuley, "Energy Is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS," *Proc. Workshop Hot Topics in Operating Systems (HotOS '01)*, pp. 59-64, 2001.
- [4] H. Zeng, C.S. Ellis, A.R. Lebeck, and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '02)*, pp. 123-132, 2002.

- [5] R. Joseph and M. Martonosi, "Run-Time Power Estimation in High Performance Microprocessors," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '01)*, pp. 135-140, 2001.
- [6] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez, "Thermal Management System for High Performance PowerPC Microprocessors," *Proc. IEEE Compcon '97*, pp. 325-330, 1997.
- [7] Q. Zhu, F.M. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," *Proc. 10th Int'l Symp. High-Performance Computer Architecture (HPCA '04)*, pp. 118-129, 2004.
- [8] L. Benini and G.D. Micheli, "System-Level Power Optimization: Techniques and Tools," *ACM Trans. Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 115-192, Apr. 2000.
- [9] F. Douglass, P. Krishnan, and B. Bershad, "Adaptive Disk Spin-Down Policies for Mobile Computers," *Proc. USENIX Symp. Mobile and Location-Independent Computing*, pp. 121-137, 1995.
- [10] C.-H. Hwang and A.C.-H. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *ACM Trans. Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 226-241, Apr. 2000.
- [11] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G.D. Micheli, "Dynamic Power Management for Nonstationary Service Requests," *IEEE Trans. Computers*, vol. 51, no. 11, pp. 1345-1361, Nov. 2002.
- [12] N. Pettis, J. Ridenour, and Y.-H. Lu, "Automatic Run-Time Selection of Power Policies for Operating Systems," *Proc. Design, Automation and Test in Europe (DATE '06)*, pp. 508-513, 2006.
- [13] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki, "Competitive Randomized Algorithms for Nonuniform Problems," *Algorithmica*, vol. 11, no. 6, pp. 542-571, June 1994.
- [14] L. Benini, A. Bogliolo, G.A. Paleologo, and G.D. Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813-833, June 1999.
- [15] Q. Qiu, Q. Wu, and M. Pedram, "Dynamic Power Management of Complex Systems Using Generalized Stochastic Petri Nets," *Proc. 37th Design Automation Conf. (DAC '00)*, pp. 352-356, 2000.
- [16] Z. Ren, B.H. Krogh, and R. Marculescu, "Hierarchical Adaptive Dynamic Power Management," *IEEE Trans. Computers*, vol. 54, no. 4, pp. 409-420, Apr. 2005.
- [17] T. Simunic, L. Benini, P. Glynn, and G.D. Micheli, "Dynamic Power Management for Portable Systems," *Proc. ACM MobiCom '00*, pp. 11-19, 2000.
- [18] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-Level Power Management for Dense Blade Servers," *Proc. 33rd Ann. Int'l Symp. Computer Architecture (ISCA '06)*, pp. 66-77, 2006.
- [19] C. Rusu, A. Ferreira, C. Scordino, A. Watson, R. Melhem, and D. Mosse, "Energy-Efficient Real-Time Heterogeneous Server Clusters," *Proc. 12th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '06)*, pp. 418-428, 2006.
- [20] C. Xian and Y.-H. Lu, "Dynamic Voltage Scaling for Multitasking Real-Time Systems with Uncertain Execution Time," *Proc. 16th ACM Great Lakes Symp. VLSI (GLSVLSI '06)*, pp. 392-397, 2006.
- [21] F. Zhang and S.T. Chanson, "Power-Aware Processor Scheduling under Average Delay Constraints," *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '05)*, pp. 202-212, 2005.
- [22] L. Cai and Y.-H. Lu, "Joint Power Management of Memory and Disk," *Proc. Design, Automation and Test in Europe (DATE '05)*, pp. 86-91, 2005.
- [23] H. Huang, P. Pillai, and K.G. Shin, "Design and Implementation of Power-Aware Virtual Memory," *Proc. USENIX Ann. Technical Conf.*, pp. 57-70, 2003.
- [24] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic Tracking of Page Miss Ratio Curve for Memory Management," *Proc. 11th ACM Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '04)*, pp. 177-188, 2004.
- [25] *Advanced Configuration Power Interface*, <http://www.acpi.info>, 2008.
- [26] *Power Policy Configuration and Deployment in Windows Vista*, Microsoft Corp., http://www.microsoft.com/whdc/system/pnpwvr/PMpolicy_Vista.msp, Oct. 2006.
- [27] D. Brownell, *Linux Kernel 2.6.17 Source: Documentation/power/devices.txt*, <http://www.kernel.org>, July 2006.
- [28] L. Cai and Y.-H. Lu, "PowerReduction of Multiple Disks Using Dynamic Cache Resizing and SpeedControl," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '06)*, pp. 186-190, 2006.
- [29] E. Pinheiro and R. Bianchini, "Energy Conservation Techniques for Disk Array-Based Servers," *Proc. Int'l Conf. Supercomputing (ICS '04)*, pp. 68-78, 2004.
- [30] N. Pettis and Y.-H. Lu, "Implementation Guides for a Homogeneous Architecture for Power Policy Integration in Operating Systems," Technical Report ECE07-10, School of Electrical and Computer Eng., Purdue Univ., 2007.
- [31] E.V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," *Proc. Int'l Conf. Supercomputing (ICS '03)*, pp. 86-97, 2003.
- [32] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: Helping Disk Arrays Sleep through the Winter," *Proc. ACM Symp. Operating Systems Principles (SOSP '05)*, pp. 177-190, 2005.
- [33] O. Celebican, T.S. Rosing, and V.J. Mooney III, "Energy Estimation of Peripheral Devices in Embedded Systems," *Proc. 14th ACM Great Lakes Symp. VLSI (GLSVLSI '04)*, pp. 430-435, 2004.
- [34] T.L. Cignetti, K. Komarov, and C.S. Ellis, "Energy Estimation Tools for the Palm," *Proc. ACM Int'l Workshop Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '00)*, pp. 96-103, 2000.
- [35] K.I. Farkas, J. Flinn, G. Back, D. Grunwald, and J.M. Anderson, "Quantifying the Energy Consumption of a Pocket Computer and a Java Virtual Machine," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems (Sigmetrics '00)*, pp. 252-263, 2000.
- [36] T. Li and L.K. John, "Run-Time Modeling and Estimation of Operating System Power Consumption," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems (Sigmetrics '03)*, pp. 160-171, 2003.
- [37] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *Computer*, vol. 27, no. 3, pp. 17-28, Mar. 1994.
- [38] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang, "Modeling Hard-Disk Power Consumption," *Proc. Second USENIX Conf. File and Storage Technologies (FAST '03)*, pp. 217-230, 2003.
- [39] J. Levon and P. Elie, *OProfile*, <http://oprofile.sourceforge.net>, 2008.
- [40] *Introduction to Plug and Play and Power Management in the Windows Driver Foundation*, Microsoft Corp., www.microsoft.com/whdc/driver/wdf/WDF_pnpPower.msp, Mar. 2006.
- [41] Y. Xie and D. Engler, "Using Redundancies to Find Errors," *IEEE Trans. Software Eng.*, vol. 29, no. 10, pp. 915-928, Oct. 2003.

Nathaniel (Eddie) Pettis received the BSEE degree from the University of Arkansas, Fayetteville, in 2002 and the PhD degree in computer engineering from Purdue University in 2008. He is currently a platforms engineer with Google, Mountain View, California. His research interests include operating systems-directed power management and embedded systems. He was named a finalist for the Choras Top Doctoral Student Award. He is a member of the IEEE.

Yung-Hsiang Lu received the PhD degree in electrical engineering from Stanford University, California in 2002. He is an associate professor in the School of Electrical and Computer Engineering, and by courtesy, the Department of Computer Science, Purdue University, West Lafayette, Indiana. In 2004, he obtained an NSF Career Award for studying energy management by operating systems. His research interests include energy management for computer systems, mobile robots, sensor networks, and image processing. He is a senior member of the IEEE.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.