Program Counter-Based Prediction Techniques for Dynamic Power Management

Chris Gniady, *Member*, *IEEE*, Ali R. Butt, *Member*, *IEEE*,
Y. Charlie Hu, *Member*, *IEEE*, and Yung-Hsiang Lu, *Member*, *IEEE*

Abstract—Reducing energy consumption has become one of the major challenges in designing future computing systems. This paper proposes a novel idea of using program counters to predict I/O activities in the operating system. It presents a complete design of Program-Counter Access Predictor (PCAP) that dynamically learns the access patterns of applications and predicts when an I/O device can be shut down to save energy. PCAP uses path-based correlation to observe a particular sequence of program counters leading to each idle period and predicts future occurrences of that idle period. PCAP differs from previously proposed shutdown predictors in its ability to: 1) correlate I/O operations to particular behavior of the applications and users, 2) carry prediction information across multiple executions of the applications, and 3) attain higher energy savings while incurring lower mispredictions. We perform an extensive evaluation study of PCAP using a detailed trace-driven simulation and an actual Linux implementation. Our results show that PCAP achieves lower average mispredictions and higher energy savings than the simple timeout scheme and the state-of-the-art Learning Tree scheme.

Index Terms—Energy-aware systems, hardware/software interfaces, storage management.

1 Introduction

Reducing energy consumption has become one of the most important challenges in designing future computing systems. While Moore's Law provides a steady reduction in power consumption per operation, increasing demand for higher performance, versatile functionalities, and better user interfaces has been raising power consumption faster than the reduction from semiconductor technology. Today, many computers are mobile, using batteries with limited capacity. Meanwhile, users expect wireless network connections, high-quality video and audio, large storage space, and so on. Efficient power management [35] will remain a major challenge in computer system design for the foreseeable future.

In this paper, we focus on reducing the energy consumption of hard disks, but the idea can be applied to other I/O devices such as wireless network interfaces. Many I/O devices are not always needed. For example, a hard disk drive is idle when all needed data reside in memory. When an I/O device is idle, it can be turned off (also called shut down) to reduce energy consumption in the system. When the device is needed later, it is turned on. This is called *dynamic power management*. Unfortunately, there are overheads to shutting down and turning on an I/O device. For example, a hard disk needs to spin up its platters. Because of the substantial overhead, a device should be shut down only if it will be idle for a period of time long enough to compensate for the overhead. If there were no overhead, power management would have been a trivial problem; a device could be shut down whenever it was idle. The critical issue in power management is to accurately predict the length of future idle periods and determine whether to shut down a device.

We propose a new mechanism, Program-Counter Access Predictor (PCAP), for dynamic power management. The idea is motivated by recent innovations in branch prediction for high-performance processors. Sequences of I/O operations are invoked by a certain group of instructions within an application. Therefore, the predictor can observe what current I/O operation is being performed and predict the outcome based on previous experiences with that particular I/O operation. The context of each I/O operation is recorded using the sequence of program counters (PCs) that precede the particular I/O. If the same sequence of PCs is repeated in the same context and was previously followed by a long idle period, then our method predicts a long idle period and shuts down the disk.

Compared with previously proposed shutdown predictors, PCAP has two major advantages. First, it uses program counters to correlate the I/O operations of each program. No information aggregation is adopted; hence, the information is exact. Second, it allows continuous improvement through multiple invocations of the same program. This is possible because the program counters that create a particular I/O operation remain the same in different executions. These two advantages are unavailable in any of the existing methods. Because of the precise information, our method is able to attain better energy savings with very few mispredictions.

We have performed an extensive evaluation study of PCAP using a detailed trace-driven simulation and an actual Linux implementation. Our results show that PCAP achieves lower average mispredictions and higher energy savings than the simple timeout scheme and the state-of-the-art Learning Tree scheme.

- *C. Gniady is with the Department of Computer Science, University of Arizona, Gould-Simpson Building, 1040 E. 4th Street, PO Box 210077, Tucson, AZ 85721-0077. E-mail: gniady@cs.arizona.edu.*
- *A.R. Butt, Y.C. Hu, and Y.-H. Lu are with the School of Electrical and Computer Engineering, Purdue University, 465 Northwestern Ave., West Lafayette, IN 47907. E-mail: {butta, ychu, yunglu}@purdue.edu.*

Manuscript received 3 Apr. 2005; revised 7 Nov. 2005; accepted 7 Dec. 2005; published online 21 Apr. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0093-0405.

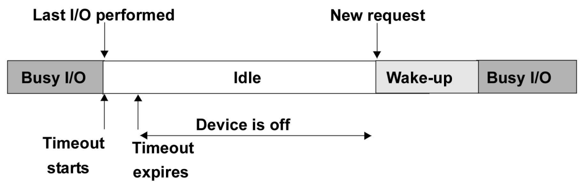

Fig. 1. Anatomy of an idle period.

The rest of the paper is organized as follows: Section 2 reviews current energy saving techniques and program counter-based prediction techniques in architecture design. Sections 3, 4, and 5 present the basic design, various optimizations, and the global predictor of PCAP. In Section 6, we present the experimental setup and a comparison of PCAP with previous prediction schemes via trace-driven simulations. In Section 7, an implementation of PCAP in Linux is evaluated. Finally, Section 8 draws concluding remarks.

2 Background

2.1 A Taxonomy of Current Predictors for Power Management

Many computers use power management to reduce energy consumption. Since the early 1990s, manufacturers have been recommending spinning down hard disks after some period of idleness [13], [20]. The simple timeout mechanism has gained wide popularity and is currently implemented in many operating systems. Fig. 1 shows an idle period divided into two intervals. When a device becomes idle, a timer starts. In the first interval, the device remains on. This interval ends when the timer expires. The device is shut down and “sleeps” during the second interval until a new request arrives. If a request arrives during the first interval, the device does not enter the second interval. This approach does not save energy during the first interval, but saves energy during the second interval.

Disks require more energy to accelerate the platters during a spin-up than during the idle state. To gain energy savings, the time in the idle state has to be long enough to offset the extra energy needed during the shutdown and spin-up sequence. This time is commonly referred to as the *breakeven time* and is usually on the order of a few seconds. The device-off time in Fig. 1 has to be larger than the breakeven time to produce any energy savings. A mispredicted shutdown results in more energy being consumed than saved. Karlin et al. [25] suggested using a component’s parameters to determine the timeout value. Their approach produced 2-competitive energy savings if the only available information was the sequence of requests from all processes. In practice, to prevent shutdowns that affects the user perceived responsiveness of the machine, the timeout period is usually set to tens of minutes. While the user is working, the long timeout intervals keep the disk in the active state, consuming energy but providing better performance. Portable computers, on the other hand, are usually either continuously used or turned off when not in use. Therefore, long timeout intervals do not produce significant energy savings in portable computers.

To address the energy savings in portable computers and further exploit opportunities for energy savings in desktop computers, dynamic predictors have been proposed based on the premise that a history of events is likely to repeat in the future due to the repetitive behavior of the applications [48]. In [49], Srivastava et al. observed that a long idle period often follows a short busy period and suggested that the length of an idle period could be predicted by the length of the previous busy period. Learning Tree [11] is the first attempt to adapt branch prediction techniques for energy management. Fig. 2 shows an example of some repetitive behavior of idle periods. Learning Tree discretizes the idle periods and uses the patterns or history of idle periods to make prediction. In Fig. 2, Learning Tree first learns that the occurrence of two idle periods shorter than the breakeven time is followed by a long idle period. If the two short idle periods occur again, they trigger the prediction of a long idle time. To reduce mispredictions, Learning Tree uses sliding window filters that filter mispredictions closely followed by an I/O operation. The sliding window filter can be applied to all dynamics predictors to prevent them from issuing a shutdown for I/O operations occurring closely together. Hwang and Wu [21] observed that the length of an idle period could be predicted using a weighted average of the predicted and the actual lengths of the previous idle period. Some other researchers suggested dynamically adjusting the timeout interval [14], [17]. These methods use feedback to enlarge or to reduce the timeout based on whether the previous prediction is correct. If it is correct, the timeout is reduced; otherwise, it is enlarged.

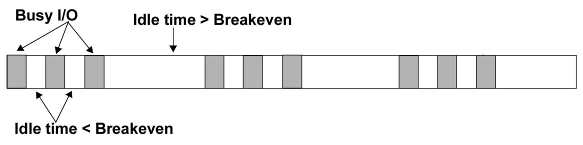

Fig. 2. Repetitive behavior of I/O accesses.

Stochastic modeling [3], [10], [42], [47] has also been used to model the trace behavior and predict the idle period based on the model parameters. In these approaches, I/O requests are considered as a stochastic process. Benini et al. [3] used stationary discrete-time Markov processes to model the arrival of I/O operations. Using this model, they obtained the optimal probability to shut down a device for achieving optimal energy saving. Chung et al. [10] extended the method and considered nonstationary accesses. Their method precomputes the optimal solutions for several I/O probabilities. At runtime, the power manager estimates the current probability and interpolates from the precomputed solutions. Qiu and Pedram [42] used continuous-time Markov models and event-triggering so the power manager would not have to periodically reevaluate whether to shut down a device. Simunic et al. [47] suggested adding timeout to continuous-time Markov models so that a device would eventually be shut down if the device was idle continuously.

A detailed study and evaluation of various predictors was presented in [29] with the following conclusions: 1) Timeout predictors offer good accuracy but waiting for timeout to expire consumes energy; 2) dynamic prediction shuts down the device immediately but had, so far, much lower accuracies than the simple timeout prediction; 3) stochastic methods usually require offline preprocessing and are more difficult to implement and problems may arise if the workload changes [10]. Application controlled power management [15], [19], [30], [51] has much better potential for reducing energy consumption. However, the technique places the burden of inserting power management directives on the programmers and requires the existing applications be modified before they can benefit from the energy management. Runtime adaptability of dynamic predictors provides an excellent platform for the design of advanced shutdown predictors. In this paper, we adopt sophisticated branch prediction techniques for energy management.

In many applications, devices do not remain idle long enough to shut down. Subsequently, research has been conducted to extend idle periods and facilitate shut down. Bertozzi et al. [4] study a shutdown method with data buffering for a network interface card. In their work, all network packets are inserted into a buffer while the network card remains in a low-power state. When the buffer reaches capacity, the network card is awakened, and all data in the buffer is transmitted. When the buffer is emptied, the network card returns to a low-power state. Their simulation results show the minimum buffer size as a function of the network bandwidth and indicate that buffers should be as large as possible. By using large buffers, the awakening overhead energy is amortized across a longer interval, increasing power savings. Ramachandran and Jacome [43] employ dedicated caches with application partitioning to produce idle periods between memory operations. Streaming data is prefetched into a dedicated cache; frequently used constants and variables are stored in a scratchpad memory to limit the number of external memory references.

Some applications do not lend themselves to shutdown policies because the amount of idleness is very low and cannot be extended due to timing constraints. Power-aware scheduling arranges the execution time of tasks to keep processors at low-power states (i.e., lower frequencies) while meeting all timing constraints. Ishihara and Yasuura [23] use integer programming for scaling discrete voltage levels. Their method adjusts the processors' performance levels to accomplish all tasks before their deadlines with the minimum energy. Luo and Jha [31] minimize processor frequencies by constructing task graphs and finding paths that minimize the ratio of slack time to worst-case execution time. Some methods use data buffers to create and prolong idle periods. For example, Im et al. [22] use buffers to exploit the slack time when multimedia applications do not use the worst-case execution time. When the processor has slack time, the processor fills the buffers so that the processor's frequency and voltage can be scaled down later. Weisel et al. [50] schedule processes to create idle periods for IO components. Pouwelse et al. [40] and Schmitz et al. [45] present scheduling techniques so that the processors' frequencies can be scaled down without missing deadlines.

Some methods have been developed to manage the energy of whole machines. For example, Neugebauer and McAuley [36] treat energy as a resource like CPU time or memory allocation. They account for the energy consumed by each process as a metric to offer the desired quality-of-service. Chase et al. [8] propose an economic model where processes bid for energy to accomplish their tasks. Zeng et al. [53] propose an energy budget for each process; the scheduler selects a process according to its remaining budget.

2.2 Program Counter-Based Prediction in Hardware

History-based prediction techniques exploit the principle that most programs exhibit certain degrees of repetitive behavior. For example, subroutines within the application are called multiple times and loops are written to process large amounts of data. The challenge in making an accurate prediction is to link the past behavior (event) to its future reoccurrence. In particular, predictors need the program context of past events so that future events that are about to occur in the same context can be identified. The more accurate the context information that the predictor has about the past and future events is, the more accurate the prediction it can make about future program behavior is.

A key observation made in computer architecture is that a particular sequence of instructions usually performs a very unique task and seldom changes behavior and that program instructions provide a highly effective means of recording the context of program behavior. Since the instructions are uniquely described by their program counters (PCs), which specify the locations of instructions in memory, PCs offer a convenient way of recording the program context.

One of the earliest predictors to take advantage of information provided by PCs is branch prediction [48]. In fact, branch prediction techniques have been so successful in eliminating latencies associated with branch resolution that they are implemented in every modern processor. The PC of a branch instruction uniquely identifies the branch in the program and is associated with a particular behavior, for example, to take or not to take the branch. Branch prediction techniques correlate the past behavior of a branch instruction and predict its future behavior upon encountering the same instruction. To further improve the accuracy of branch predictors, the execution path which consists of the addresses of basic blocks taken before a branch instruction was used as additional context for the branching behavior [48], [34], [52]. These techniques are called *path-based predictions*.

The success of using the program counter in branch prediction was observed and PC information has been widely used in other predictor design in computer architecture. Numerous PC-based predictors have been proposed to optimize energy [2], [41], cache management [26], [27], and memory prefetching [46], [5], [16], [39], [1], [24]. Program counters have been used to accurately predict the instruction behavior in the processor's pipeline, which allows the hardware to apply power reduction techniques at the right time to minimize the impact on performance [41], [2]. In Last Touch Predictor [27], [26], PCs are used to predict which data will not be used by the processor again and free up the cache for storing or prefetching more relevant data. In PC-based prefetch predictors [46], [5], [16], [39], [1], [24], a set of memory addresses or patterns is linked to a particular PC and the next set of data is prefetched when that PC is encountered again.

PC-based techniques have also been used to improve processor performance by predicting instruction behavior in the processor pipeline [44], [9] for better utilization of resources with fewer conflicts, as well as to predict data movement in multiprocessors [32], [26] to reduce communication latencies in multiprocessor systems.

Despite their tremendous success in architecture design, PC-based techniques have not been explored in operating systems design. In this paper, we consider a path-based prediction [34] for energy management in the operating system. Previously, the path-based prediction was demonstrated to work well in predicting a "last touch" to the cache block [26], in which the path contains program counters of cache accesses performed before the invalidation of a cache block. If the same path is encountered again, the cache block is invalidated immediately. We can immediately draw similarities between path-based prediction for memory references and I/O events leading to a last I/O before an idle period. This close relation suggests that path-based prediction will work equally well in predicting the I/O behavior in the operating system.

2.3 Energy Conservation in Data Servers

The focus of this paper is on energy conservation of a single disk in a laptop or desktop environment. There is a large body of work on energy conservation in data servers which involve multiple disks [7], [12], [18], [38], [54], [55]. One approach is to exploit the fact that server workloads exhibit wide variations in intensity over time to conserve energy. Along these lines, Carrera et al. [7] and Gurumurthi et al. [18] considered multispeed disks. These papers showed that significant energy savings can be accrued by adjusting the disk rotational speeds according to the load imposed on the disks. Carrera et al. also show that a combination of laptop and SCSI disks can be even more beneficial in terms of energy, but only for overprovisioned servers.

The other approach is to increase disk idle times so that disks can be sent to low-power modes and be kept there for longer periods of time. Zhu et al. [54] considered storage cache replacement techniques that selectively keep blocks from certain disks in the main memory cache to increase their idle times. Recently, Zhu et al. [55] studied a novel energy-aware storage cache replacement policy in which dynamically adjusted memory partitions are used for caching data from different disks. These works do not involve data movement, which could provide further energy savings.

In contrast, Colarelli and Grunwald [12] proposed the Massive Array of Idle Disks (MAID), in which data are copied to "cache disks" to increase idle times at the regular (noncache) disks. More recently, Pinheiro and Bianchini [38] demonstrated that relying on file popularity and migration, as in their Popular Data Concentration (PDC) technique, produces more robust energy savings than in the MAID approach. Their idea was to migrate the popular data to a subset of the disks so that other disks would become idle for longer periods.

3 PCAP

We propose Program Counter-based Access Predictor (PCAP), a new dynamic prediction method that can accurately predict idle periods. The key idea behind PCAP is that there is a strong correlation between a sequence of I/O operations invoked by a particular sequence of instructions within an application and the immediately following idle period. To take advantage of the repetitive functions performed by applications, PCAP extracts the program context by recording each sequence of PCs that have triggered I/O operations before a long idle period and predicts future idle periods based on previous experiences. Thus, PCAP differs from existing methods that lack the detailed program context of I/O operations.

3.1 Path-Based Prediction

A naive implementation of PCAP, motivated by a hardware one-bit branch predictor, would only record a single PC that causes an I/O followed by an idle period. If this PC is encountered again, it triggers a prediction that this is an I/O before an idle period. While this simple implementation is fairly accurate in predicting the idle periods, it is unable to accurately distinguish between the periods that are longer or shorter than the breakeven time. For example, an application reads multiple files in a loop and only the last read is followed by an idle time that is longer than the breakeven time. Using a single PC would result in the misprediction of an idle period after each file was read at the beginning of each loop iteration. Moreover, at the end of a loop iteration, the single PC predictor would not predict an idle time. The same scenario occurs when a user consecutively opens multiple files upon starting an editor.

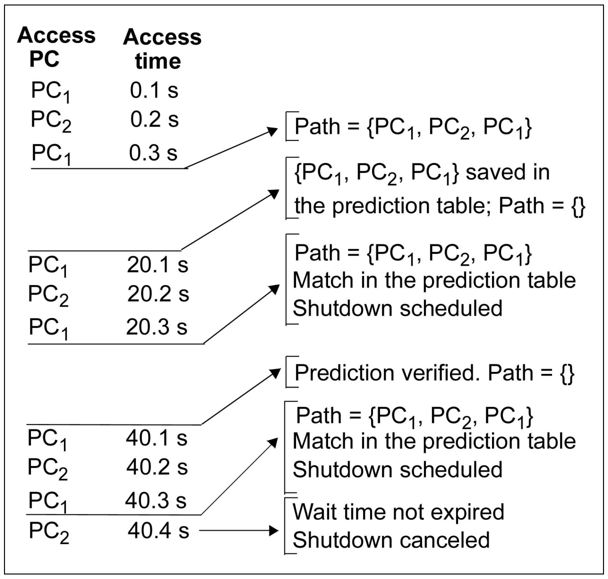
Access PC
Access time
PC_1 0.1 s
PC_2 0.2 s
PC_1 0.3 s
Path = {PC_1, PC_2, PC_1}
{PC_1, PC_2, PC_1} saved in the prediction table; Path = {}
PC_1 20.1 s
PC_2 20.2 s
PC_1 20.3 s
Path = {PC_1, PC_2, PC_1} Match in the prediction table Shutdown scheduled
Prediction verified. Path = {}
PC_1 40.1 s
PC_2 40.2 s
PC_1 40.3 s
PC_2 40.4 s
Path = {PC_1, PC_2, PC_1} Match in the prediction table Shutdown scheduled
Wait time not expired Shutdown canceled

Fig. 3. A prediction example showing the program counters that initiate I/O, time of access, and prediction steps undertaken by PCAP.

To address these problems, PCAP records a *path*: a sequence of I/O triggering PCs that starts after a hard disk idle period and leads to the next idle period. As a result, PCAP can distinguish different paths of execution and identify a particular path that the application currently follows. The path of execution leading to the current disk access will allow PCAP to identify the context of the I/O operation, resulting in a more accurate prediction. Previously, path-based prediction was used to increase the accuracy of branch prediction [34] and was later successfully used in predicting cache block eviction [26].

Fig. 3 shows an example of I/O operations made by an application. The leftmost column lists the program counters that initiate I/O operations. The middle column shows the time when an I/O operation occurs. The right column shows the prediction steps undertaken by PCAP. The application generates three sequences of I/O operations. Within each sequence, the accesses are 0.1 seconds apart, keeping the disk spinning. During the first sequence, the PC of each I/O operation is retrieved and stored as a part of a path which consists of $\{\mathrm{PC}_1, \mathrm{PC}_2, \mathrm{PC}_1\}$. At that point, PCAP encounters a long 20 second interval which presents the opportunity to save energy. This is the first time that PCAP encounters such a sequence of PCs, therefore the sequence does not trigger a prediction. However, the path is stored in the prediction table for future predictions. The second occurrence of $\{\mathrm{PC}_1, \mathrm{PC}_2, \mathrm{PC}_1\}$ triggers the prediction of an idle period and the disk is shut down. The example also shows the third sequence of $\{\mathrm{PC}_1, \mathrm{PC}_2, \mathrm{PC}_1\}$ that is immediately followed by PC_2. In this case, the misprediction will occur if there is no additional information present.

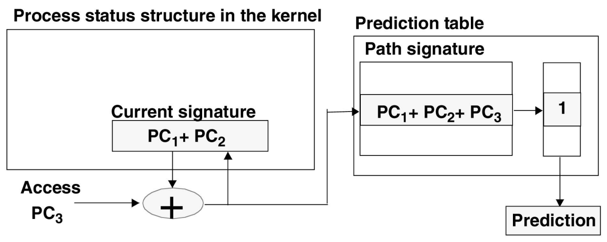

Fig. 4. Basic design of PCAP.

3.2 Basic Design

So far, we have discussed predicting idle periods based on a path of I/O triggering PCs executed by an application. The path can be arbitrarily long and, therefore, the storage and comparison can be difficult to implement efficiently. In our implementation, we encode the path by arithmetically adding the PCs in the path, as previously suggested in [26] in the context of predicting cache accesses. Such an encoded path results in a 4-byte variable, called a *path signature* in the rest of the paper. For example, a path $\{\mathrm{PC}_1, \mathrm{PC}_2, \mathrm{PC}_1\}$ from Fig. 3 is encoded as $\mathrm{PC}_1 + \mathrm{PC}_2 + \mathrm{PC}_1$. The encoding minimizes the storage requirements of PCAP and provides a quick comparison between the current path signature and the path signature in the prediction table. Such encoding introduces the possibility of two different paths resulting in the same path signature. For example, path $\{\mathrm{PC}_1, \mathrm{PC}_2, \mathrm{PC}_1\}$ is different from path $\{\mathrm{PC}_1, \mathrm{PC}_1, \mathrm{PC}_2\}$, but they will result in the same path signature. In our experiments, this signature aliasing did not occur. Therefore, we do not explore alternative encodings.

Fig. 4 illustrates runtime encoding of the path and prediction table lookup. Each process maintains its own 4-byte current signature variable in the kernel process status structure. After a period of idle time greater than the breakeven time, the current signature variable in the current process is reset to the PC of the first I/O operation. For each subsequent I/O operation, the PC that triggers the I/O is added to the current signature variable. After each update of the current signature variable, PCAP uses the signature to look up the prediction in the prediction table. If a signature match is found between the current signature and a path signature in the prediction table, PCAP predicts a long idle period and shuts down the disk. If a signature match is not found, the prediction of "no idle" is implied and the disk remains turned on. If PCAP encounters an idle period longer than the breakeven time and the current signature does not match any of the prediction table entries, PCAP records that path signature in the prediction table. After PCAP learns the new path signature, it will use the new path signature for future predictions.

3.3 Obtaining Signature PCs of I/O Operations

In calculating a path signature leading to an idle period, instead of obtaining a single PC of the function call from the application that invokes each I/O operation, PCAP actually obtains a *signature PC* which is the sum of the sequence of PCs encountered in going through multiple levels of wrappers before reaching the actual I/O system call. Wrapper functions are commonly used to simplify programming by abstracting the details of accessing a particular file structure. For example, the call graph in

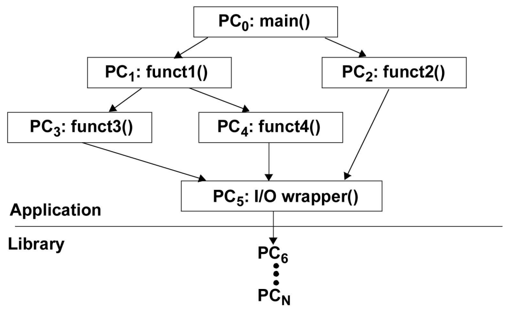

Fig. 5. Example function call graph.

Fig. 5 shows that Functions 2, 3, and 4 will use the same PC from the wrapper function for I/O operations. Therefore, the PC that invokes the I/O within the wrapper cannot differentiate the behavior of different caller functions. To obtain a unique characterization of the access PC for general applications, PCAP traverses multiple function stacks in the application. The PCs obtained during the stack frame traversal are summed together to obtain a unique identifier as the *signature PC* of the I/O operation. In the studied applications, traversal of only two additional frames in the application space provided sufficient information to PCAP. The signature PCs of the I/O operations that lead to an idle period are then encoded to calculate the path signature for that idle period.

4 PCAP Optimizations

The basic design of PCAP is able to retrieve and use program context; as a result, it can achieve more energy savings while incurring few mispredictions. In this section, we discuss adaptation of branch prediction mechanisms as well as basic timeout mechanisms to further reduce the mispredictions and improve the energy savings in PCAP.

4.1 Reducing Mispredictions

The path-based prediction method in PCAP uses the context of execution in making more accurate predictions, but it can still cause mispredictions. PCAP, as any other predictor derived from path-based prediction, inherits the possibility of *subpath aliasing*. Subpath aliasing occurs when one path of PCs is a prefix sequence within a longer path of PCs. The last sequence of accesses in Fig. 3 shows the occurrence of subpath aliasing; the path $\{\mathrm{PC}_1, \mathrm{PC}_2, \mathrm{PC}_1\}$ is a subpath of $\{\mathrm{PC}_1, \mathrm{PC}_2, \mathrm{PC}_1, \mathrm{PC}_2\}$. In this case, the misprediction occurs when the prefix path of the longer path is encountered. One example of such a scenario is when the user opens a file, performs "save as" to a different file, opens another file, and edits it for some period of time. The same sequence is followed later, but, instead of editing the second file, the user also performs "save as." Another example appears with an Internet browser where some pages require loading additional libraries (additional I/Os) to decode the multimedia context and some do not. In the following, we discuss two optimization techniques that reduce the mispredictions due to subpath aliasing

4.1.1 *Sliding Wait-Window*

To reduce the mispredictions due to subpath aliasing, PCAP uses a *sliding wait-window* filter before shutting down

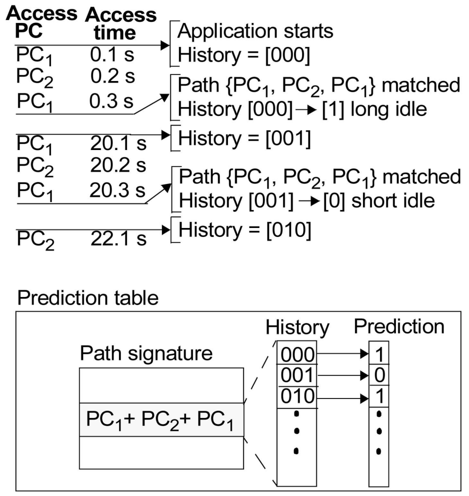

Fig. 6. Example of PCAP prediction assisted with idle period history.

the disk. In Fig. 3, the occurrence of the third sequence $\{PC_1, PC_2, PC_1\}$ will result in a shutdown prediction. After the prediction is made, the predictor waits for a sliding wait-window to pass before shutting down the disk. If, during this interval, PC_2 arrives, the prediction is ignored and the path collection is continued without any interruption. On the other hand, if there is no access during the wait-window, the disk is shut down.

4.1.2 *Idle Period History and File Descriptors*

The sliding wait-window may not be able to eliminate all mispredictions caused by subpath aliasing. To further reduce mispredictions due to subpath aliasing, we provide PCAP with additional information about the context which will help PCAP in distinguishing different paths. We propose two sources of additional information: history of idle periods and the file descriptor of an I/O operation. These sources are orthogonal and can be implemented together to further improve the accuracy of PCAP.

History-based prediction is drawn from the wealth of optimizations proposed for branch predictors [48]. We incorporate the history of idle periods in PCAP as follows: Any idle period longer than the wait-window and shorter than the breakeven time is recorded as 0 in the idle bit-vector. Any period that is longer than the breakeven time is recorded as 1. Intervals shorter than the wait-window are not included since they are filtered at the runtime. The current path of PCs and the history bit-vectors are maintained concurrently for each running process and used together in training and predicting. The shutdown is issued only if the current path and the current idle bit-vector match a particular entry in the prediction table.

Fig. 6 shows an example of prediction in PCAP using the history of idle periods. As in Fig. 3, the leftmost column lists the program counters that initiate I/O operations, the middle column shows the time when an I/O operation occurs, and the right column shows the prediction steps undertaken by PCAP. We assume at the beginning of the sequence of I/O operations, PCAP has been trained and the prediction table is shown in the lower part of Fig. 6. The 3-bit per path history is initialized to [000]. Once the path $\{PC_1, PC_2, PC_1\}$ is matched in the prediction table, the corresponding history is looked up and the prediction obtained. In this case, the history of [000] in the prediction table predicts a long idle period [1] and the disk is shut down. Once the long idle period is finished, the current history is updated to [001]. With this history, the second occurrence of the path $\{PC_1, PC_2, PC_1\}$ is predicted to be followed by a short idle period [0]. This implies that the path $\{PC_1, PC_2, PC_1\}$ is the subpath of some longer path, i.e., a case of subpath aliasing, and the disk is not shut down. At the end of the idle period, the history is updated to [010]. In this simple example, idle period history information helps PCAP in identifying subpath aliasing and making the correct predictions.

Inclusion of file descriptors into the prediction table entries is motivated by studies in [26], where the authors use the address of the cache block to aid the predictor in differentiating cache blocks that exhibit subpath aliasing. A straightforward adoption would use the location of accessed files on the disk. However, inclusion of file locations makes the prediction table size dependent on the I/O footprint of the application and the table size can grow substantially [26]. Moreover, an application can potentially open different files in different executions, requiring the predictor to retrain every time a new file is open. File descriptors alone, on the other hand, show less variability and provide related context because file descriptors are often assigned based on some user behavior.

4.2 Reusing Prediction Tables

Path-based prediction requires extensive training to populate the prediction table. To reduce the delay in training, we propose reusing the prediction tables across multiple executions of the same application. PCAP uses training based on process ID; it associates the prediction table with a particular application. Once the application exits, the trained prediction table is saved in the application initialization file, which most applications already have. When the application starts again, the prediction table is loaded by reading the initialization file.

The uniqueness of PCs allows the prediction table to be carried across application executions. However, PC addresses may change due to recompilation or dynamically loadable modules. In this case, PCAP will retrain based on the new code or the order of loaded modules.

4.3 Backup Predictor

There are two situations in which the energy consumption of the disk cannot be saved by the PCAP technique described so far. First, during training for a new path signature, PCAP does not make a shutdown prediction and the disk will remain spinning for the entire idle period. Second, when PCAP predicts the disk will be busy following an I/O when, in fact, the disk will be idle for a period longer than the breakeven time, the disk again will remain spinning. To reduce the impact of training and misprediction on energy savings, when PCAP is unable to match a signature or predicts not shutdown, the backup timeout predictor shuts down the disk after the timer expires. In both cases, the timeout predictor overrides the no-idle prediction from PCAP and shuts down the disk.

4.4 Alternative Low Overhead Design

A major overhead of PCAP comes from obtaining the PCs of the call instructions that trigger I/O operations. In the default design, PCAP obtains the PC during each I/O operation in order to produce a complete signature of the execution path, although it only performs the table

TABLE 1
Summary of Optimization Techniques

Optimization	Description	Benefits	Drawback
Sliding-wait window	Delays shutdown prediction	Significantly reduces mispredictions during clustered accesses, higher energy savings	The short timeout may reduce energy savings
Idle period history	Records the sequence of idle periods for a path	Reduces sub-path aliasing	Increases storage overhead and training time
File descriptors	Correlates file accesses to a PC path	Improves accuracy in cases where same path shows different behavior for multiple files	Storage overhead related to the number of files (usually very high), significantly increased training time
Table reuse	Saves prediction table for future executions	Reduces training in future executions	Overhead of storing and reloading of prediction tables
Timeout backup	Shuts down the disk when PCAP misses prediction	Provides energy savings during training in PCAP	None

lookup upon the disk access after the buffer cache miss. An alternative PCAP design which reduces the overhead of obtaining PCs is to collect the PCs of the I/Os only upon buffer cache misses. This approach also localizes changes to the kernel since a single function call will perform PC retrieval, signature calculation, prediction table update, and the prediction lookup for the current I/O. In addition, this implementation reduces the overhead of signature calculation since a significant number of I/O requests are performed in the buffer cache [6], [33] for which the PC is not retrieved.

In this design, the PCAP overhead only occurs during file cache misses and is overshadowed by the miss processing overhead and disk access latencies. However, since fewer PCs are collected, this design also leads to much sparser paths of PCs. Sparser PC paths will result in a higher likelihood of subpath aliasing and, therefore, the use of file descriptor or history information may become necessary. This additional information will complement the reduced context to maintain a comparable prediction accuracy as before. We evaluate the trade-offs in this low overhead design in Section 6.5.3.

4.5 Summary

We summarize the various design optimizations and their impact on the prediction accuracy and energy savings in Table 1. In the following, we elaborate on these design optimizations:

- **Sliding-wait window** is similar to the timeout backup predictor in that it does not present any implementation overhead. However, due to the delay in shutting down the disk, a small amount of energy consumption will be present even with a perfect prediction. In real workloads, perfect prediction is not possible and sliding-wait window offers a significant reduction in mispredictions and, consequently, a reduction in energy due to reduced unnecessary shutdowns and spin-ups. Overall, sliding-wait window improves the prediction accuracy and energy savings.
- **Idle period history** is effective in reducing the subpath aliasing. The history records the path and can differentiate between the current path being a subpart of a longer path or the longer path itself. The history, however, increases the storage overhead for each path since each path can have multiple outcomes based on the history. Furthermore, the training period is increased as PCAP is trained for each combination of a path and each of its associated histories even if the outcomes of different histories are the same. This optimization is justified in cases where limited information can result in sparse paths and significant subpath aliasing, as discussed in Section 4.4.
- **File descriptors** are similar to the idle history above in terms of the storage overhead. However, each path has a finite number of histories, while each PC path can access an unbounded number of files, making storage requirements unbounded. This can impact the memory footprint of the predictor in the implementation. Using a fixed size table to limit the number of entries trained, which is necessary in a real implementation, may limit PCAP's ability to make predictions. Furthermore, any accesses to new files will require retraining, resulting in reduced opportunities for making predictions.
- **Table reuse** is necessary for advanced predictors such as PCAP and LT. The prediction table is saved when an application exits. The cost of saving the prediction table and loading it for future executions is low as long as the table size is small.
- **Timeout backup** is effective since it provides an opportunity to save energy during the training period of the main predictor. There is little overhead associated with the implementation and the longer timeout period associated with the backup predictor does not impact the accuracy of the main predictor. Therefore, this optimization should be used in advanced predictors that require training to predict idle periods such as PCAP and LT.

From the above discussion, we conclude that timeout backup, sliding-wait window, and table reuse should be included in the base PCAP design. The other two

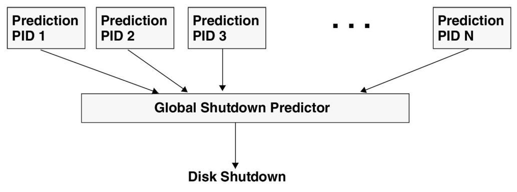

Fig. 7. Global predictor.

optimizations, using idle period history and file descriptors, involve design trade-offs which need to be carefully examined. We evaluate these two optimizations in our trace-based simulations in Section 6.

5 Global Prediction

So far, we have discussed PCAP implementation and optimization on a per-application basis. In real systems, many processes are running concurrently and some of them may be from a single application. A system-wide prediction is needed that will take into account multiple processes running concurrently. Fig. 7 presents the Global Shutdown Predictor that generates shutdown predictions by considering the input from all processes. Each process has its own private PCAP which generates local predictions as shown in Fig. 4. The Global Shutdown Predictor predicts shutdown only when the PCAP for every process in the system predicts shutdown.

PCAP for each process generates prediction only after an I/O operation. Once a prediction to shut down the disk is generated, it remains unchanged until the process performs an I/O operation that wakes up the disk. The design of the predictors guarantees that the shutdown prediction will be made for each idle period either by the primary predictor right before the idle period starts or by the backup timeout predictor after the timer expires. If PCAP is in training, the backup timeout predictor will make the prediction for that process. For example, assume that PCAPs from all processes in Fig. 7 predict shutdown and the disk is turned off. At some later time, Process 2 performs some I/O operation that wakes up the disk and its PCAP predicts the shutdown right after the access. Since other processes do not change their state, all predictions remain the same and the disk is shut down after Process 2's PCAP makes the prediction. We can observe that PCAP from the currently running process will make the last prediction and no synchronization is necessary between the waiting processes.

6 Experimental Results

In this section and the next, we evaluate the performance of PCAP. This section presents a simulation study and the next section presents an implementation study.

6.1 Experimental Setup

To evaluate the performance of PCAP and compare it to previously proposed predictors, we used a trace simulator. A detailed trace of the applications was obtained by modifying the `strace` Linux utility. The modified `strace` reads a traced process's memory and allows us to obtain the following information about the I/O operation: PC, access type, time, file descriptor, and file location on disk. In addition, we also included the time of `forks` and `exits` of the processes within the parent application. Each application was traced separately, creating an independent trace for each application.

TABLE 2
Applications and Execution Details

	Num. of	Num. of idle periods		Total
Appl.	executions	Global	Local	I/Os
mozilla	49	325	951	90843
writer	33	108	354	133016
impress	19	69	227	220455
xemacs	37	90	102	79720
nedit	29	29	29	6663
mplayer	31	51	107	512433

Table 2 shows six applications used by a user during a seven-day trace collection period. *Mozilla* is a Web browser and the user spends time reading the page content and following the links. The I/O behavior depends on the content of the page and the interests of the user. *Xemacs* and *nedit* are editors used by the user who spends most of the time thinking and typing. *Xemacs* is primarily used to create larger files and edit multiple files, while *nedit* is primarily used to quickly open correct/modify source code during compilation or bug fixes. *Nedit* does not show repetitive behavior since, once a file is modified, it is saved and *nedit* is closed. Nedit is the only application with a single process. *Writer* is a word processor from the Open Office suite [37] and the user mostly composes the text and also does some quick fixes after proofreading. *Impress* is also an Open Office application and is used to prepare presentation slides. *Mplayer* is a media player and the user usually watches a media clip and then exits the player.

Table 2 also lists how many times each application was executed and the total number of idle periods that were long enough to save energy by performing a shutdown. The local number of idle periods is the sum of idle periods that each process from the application encountered. The global number shows the idle periods observed by an application as a whole, i.e., the number of periods when all processes observed idle I/Os. Therefore, the global number is much smaller than the sum of local numbers except *nedit*.

The trace simulator simulates the multiprocess environment. It simulates different idle period predictors and collects execution statistics for each process, as well as for the entire application.

To take into account the effects of disk caching in an operating system, we implemented a file cache simulator. The simulator models the implementation of the file cache in Linux and the collected traces of I/O operations are filtered through our file cache and only cache misses are treated as actual disk accesses. The file cache size is 128 MBytes. The LRU policy is used for cache replacement and the extended timer of 10 minutes is used between cache flushes of dirty blocks. Since the studied applications did not generate a large amount of blocks, the impact of dirty data flushes was limited. The elongation of default timer and optimizations of dirty data flushes are currently being evaluated in the Linux community [28]. These optimizations will further benefit the power management.

Energy consumption and savings are calculated based on the amount of time the applications spend in a particular

TABLE 3
The States and State Transitions of the Simulated Disk

State	Power
Busy power	2.6 W
Idle power	1.3 W
Standby power	0.25 W
Sleep power	0.10 W
Breakeven energy	5.37 J
Breakeven time	5.11 sec.

state and the corresponding power consumption are listed in Table 3. These parameters correspond to a 20 GB Hitachi Travelstar IC25N020ATCS05 disk drive [29].

We start by evaluating the ability of various predictors to predict shutdowns in Sections 6.2 and 6.3. In Section 6.4, we evaluate energy savings of various predictors. In Section 6.5, we compare the effectiveness of different optimizations of PCAP for reducing mispredictions and training time.

6.2 Local Prediction Accuracy

In this section, we compare the accuracy and the ability of various predictors to predict hard disk shutdowns. In Fig. 8, we compare the timeout predictor (TP), the Learning Tree (LT) predictor, and PCAP. TP uses a 14-second timer and after the timer expires it shuts down the disk. The 14-second interval is chosen because it results in the lowest mispredictions and the best energy savings in our applications. Lower timer values would increase the number of mispredictions significantly and much longer timeout would reduce the energy savings considerably. The 14-second interval is also used for the backup timeout predictors in PCAP and LT. LT is able to manage multiple power states, but, in our study, LT manages only two states: on and off. The backup timeout predictor and the sliding wait-window mechanism are included in both LT and PCAP, allowing a direct comparison. We used one-second wait-window since it filters mispredictions in most common cases.

Fig. 8 presents the fractions of shutdowns normalized to the number of idle periods that are long enough for a shutdown to benefit energy management. The fraction of "Hit" represents the fraction of idle periods with correctly predicted shutdowns. The "Not Predicted" fraction represents missed opportunities to shut the disk down. The fraction of "Miss" corresponds to the additional shutdowns that were introduced due to misprediction. These additional

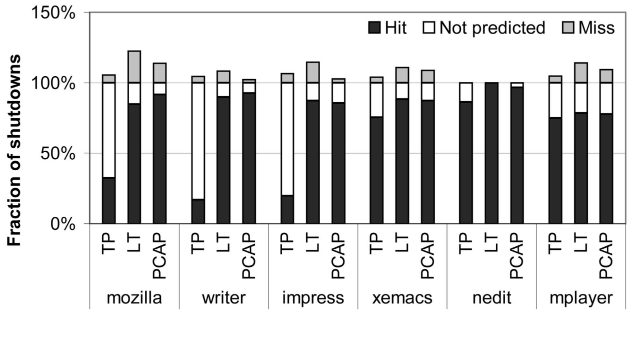

Fig. 8. Local shutdown predictor.

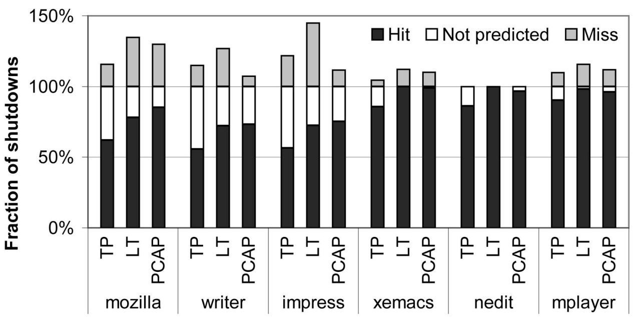

Fig. 9. Global shutdown predictor.

shutdowns occurred during the idle periods that were shorter than the breakeven time and, therefore, are not part of the idle periods shown in Table 2. However, we normalized the number of mispredictions to the number of idle periods for direct comparison in the figures in this section. It is preferred that we have a 100 percent black bar without any white or gray bars to indicate 100 percent accurate predictions. TP has the lowest coverage, 51 percent on average, and, as a result, it has the lowest number of mispredictions, 4 percent on average. Here, the *coverage* is defined as correctly predicted shutdowns as a percentage of all such opportunities. *Mozilla*, *writer*, and *impress* have multiple processes with short idle intervals. *Mozilla* is the most difficult to predict since it has many short intervals that result from the user following the links on the Web pages. The remaining applications usually have longer idle periods and TP performs better.

The wait-window makes the number of mispredictions in LT rather low for the dynamic predictor, which averages 12 percent across the applications. LT is able to correctly predict 88 percent of local shutdowns. To maximize energy savings and minimize mispredictions, we used a history length of eight in LT. Using a longer history does not improve accuracy. Using a shorter history results in more hits, but misprediction also increases.

PCAP achieves the highest average coverage by correctly predicting 89 percent of the local shutdown intervals. PCAP has slightly lower coverage in *nedit* and *mplayer* than LT, as it requires one more idle period to learn in *nedit* and two more in *mplayer*. This is because PCAP requires more training than the predictors that do not observe the application context. PCAP also improves the prediction accuracy, compared to LT, with only 6 percent mispredicted shutdowns. Compared to TP, PCAP has 38 percent higher coverage and only 2 percent more mispredictions. The mispredictions in PCAP can be significantly reduced by providing more context, as shown in Section 6.5.

6.3 Global Prediction Accuracy

The final shutdown prediction is made by the global predictor; it makes predictions based on the collection of local predictions. In the remaining sections, we will only present global prediction results.

Fig. 9 shows the final prediction results made by the Global Shutdown Predictor. Results were normalized to the number of global idle periods since only during those periods should the predictors attempt to shut down the disk. All applications except *nedit* involve multiple processes. As a result, only *nedit* has the same results as in Fig. 8. Fig. 9 follows the trends of Fig. 8 except for the following differences: First, TP achieves a much higher

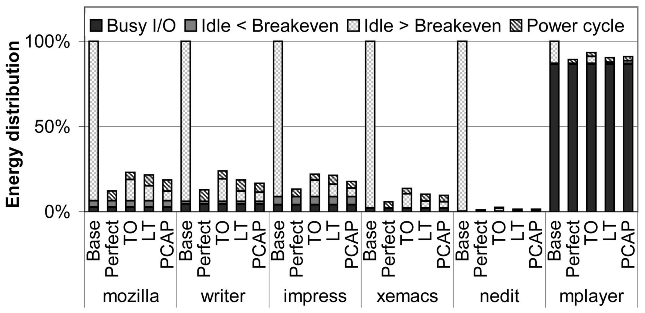

Fig. 10. Energy distribution.

percentage of hits than in Fig. 8. This can be explained by the lower number of global idle periods that the predictions are normalized against, as shown in Table 2. Second, LT and PCAP achieve a lower percentage of hits than in Fig. 8. This is caused by mixed TP (backup) and LT or PCAP as local predictors. TP requires a much larger timeout period (14 seconds) before predicting a shutdown, while LT and PCAP can make predictions immediately. Therefore, if any local predictor is using TP, the global prediction has to wait for 14 seconds before predicting a shutdown. In other words, the global predictor is coerced by the backup TP predictor into delaying making predictions. Third, all three predictors achieve a higher percentage of mispredictions. This is because the global predictor predicts shutdown only when all local predictors predict shutdown. Thus, if one local predictor mispredicts shutdown while other local predictors correctly predict shutdown, the global predictor mispredicts.

Global TP is able to shut down the disk in 73 percent of idle periods, on average, while incurring only 11 percent of mispredicted shutdowns. Global LT is more aggressive, with an average of 87 percent correct shutdowns, but also incurs an average of 22 percent mispredicted shutdowns. Global PCAP again predicts more accurately than global LT, correctly shutting down the disk during 88 percent of the idle periods, on average, while incurring only 12 percent of mispredicted shutdowns. The relative performance of the predictors across the applications remain unchanged from Fig. 8.

6.4 Energy Savings

In this section, we present a breakdown of the disk I/O operations and the ability of TP, LT, and PCAP to reduce energy consumption. Fig. 10 shows the energy consumption profile of each application. The energy consumed by each application was divided into three components: "busy I/O," "idle $<$ breakeven," and "idle $>$ breakeven." In addition, we include the "Power Cycle" fraction in the predictor results, which is the energy consumed during the shutdown and spin-up cycle for both correctly and incorrectly predicted shutdowns. The "idle $>$ breakeven" energy component is the energy consumed during the periods that are long enough to shut down the disk and save the energy.

We observe that the base system spends most of its execution in the idle I/O state. On average, 84 percent of energy is consumed during the idle I/O state and 82 percent of energy is from the intervals longer than the breakeven time. The exception is *mplayer*, which requires a continuous stream of video and, therefore, has limited idle time. *Mplayer* loads the movie into its own memory buffer and maintains the buffer full until the movie ends. At this time, the I/O activity stops and the movie finishes playing from the buffer. The idle energy in Fig. 10 corresponds to the amount of time it took to empty the 8MB buffer at the end of the movie. The idle time in the other applications depends on the user interactions. *Mozilla* loads libraries and saves temporary information every time a user opens a new Web page. Therefore, the idle time is dependent on the surfing habits of the user and the page content. The two editors, *xemacs* and *nedit*, show similar behavior since users spend more time typing and thinking than opening new files. *Writer* and *impress* are basically editors, but word processing and presentation preparation require additional libraries, such as dictionaries or graphic filters, which require more I/O time.

The ideal predictor in Fig. 10 saves all energy that comes from the idle periods that are longer than breakeven time. The energy required to turn the disk off and on is present since even the ideal predictor consumes energy during the correct shutdown and spin-up of the disk. As a result, the ideal predictor eliminates, on average, 78 percent of disk energy consumption during the execution of the applications.

TP performs well, saving, on average, 70 percent of energy in the applications, which is 8 percent away from that by the ideal predictor. Based on the analysis of [25], we can obtain two-competitive energy management, i.e., under which the total energy is at most twice that under a perfect energy manager, by setting the timeout value to the breakeven time. In this case, TP with a timeout of 5.11 seconds eliminates, on average, 71 percent of energy; however, the global mispredictions increase to 12 percent (from 11 percent) as a result of the shorter timeout.

LT is more aggressive in making predictions and saves, on average, 73 percent of energy. PCAP predictor saves, on average, 74 percent of energy, which is only 4 percent from the maximum savings possible with much lower mispredictions than LT. Misprediction rates play a very significant role in selecting the right predictor. Unnecessary shutdowns not only consume energy but can also irritate the user who has to wait for the disk to spin up.

Timeout helps LT and PCAP as a backup during training intervals. A longer timeout has an adverse effect on the energy savings in TP since TP has to wait for the timer to expire for every interval. Moreover, the energy saving-misprediction trade-off varies among applications for TP, making it even more difficult to select a value that will benefit a wide range of applications. However, LT and PCAP energy savings are only slightly affected by the timeout value since most predictions are handled by their primary predictors.

6.5 Optimizations

In this section, we first evaluate the benefits of additional context provided by idle period history and file descriptor. We then evaluate the importance of prediction table reuse.

6.5.1 *Idle Period History and File Descriptors*

The prediction accuracy is improved by providing the predictor with additional information about the context of execution. Fig. 11 compares the base version of PCAP from Fig. 9 to the base version with the addition of idle period history (PCAPh), file descriptor (PCAPf), and a combination of history and file descriptor (PCAPfh). Fig. 11 presents the results for the global predictors. Each misprediction and hit portion of the bar was split into two sections to show the contribution from the primary and backup predictors. Since there were multiple processes

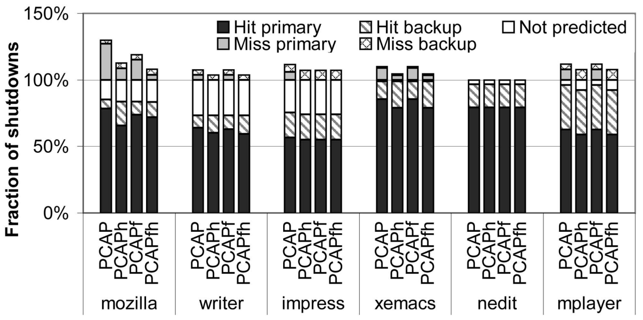

Fig. 11. Predictor optimizations.

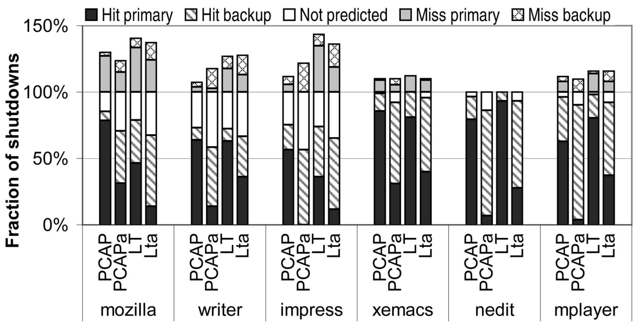

Fig. 12. Prediction table reuse.

running and making predictions concurrently, we decided to attribute the final global prediction to the predictor type (primary or backup) making the last decision before the shutdown. For example, if all processes predicted shutdown and one process is waiting for the timer to expire, this shutdown is attributed to the backup timeout predictor.

PCAP is the best performer in Fig. 9, achieving a high coverage of 88 percent at a relatively low cost of only 12 percent mispredicted shutdowns. By augmenting PC paths with the history of idle periods, we further pinpoint the location of the I/O instructions within the execution flow of the application. We have used a history length of six periods, which maximizes energy savings and minimizes the number of mispredictions. Longer history does not reduce mispredictions any further. Addition of history increases the training duration in PCAP, requiring the backup predictor to make more predictions. On average, the hit rate decreases to 86 percent, but the additional context provided by history reduces the mispredictions from an average of 12 percent to an average of 6 percent. As a result, PCAPh achieves higher coverage and fewer mispredictions than both TP and LT, as shown in Fig. 11. The impact of using history on energy savings is limited and results in well under 1 percent average change. As a result, PCAPh is still able to save 74 percent of energy at a cost of only 6 percent mispredicted shutdowns.

Mozilla is the most difficult to predict; however, PCAPh manages to reduce the misprediction rate to 13 percent as compared to 30 percent in PCAP. Thus, the additional complexity that history introduces is well justified in case of *mozilla*. The resulting total of 49 mispredicted shutdowns in 49 executions of *mozilla* should be mostly unnoticeable and should not irritate the user. Other applications have lower misprediction rates than *mozilla* and already perform well with PCAP, but are seeing 1 to 6 percent additional reduction in mispredictions.

The addition of file descriptors to the path of PCs (PCAPf) also improves PCAP's accuracy of prediction. But, since a file descriptor may be reused by multiple files, the accuracy is not as good as in PCAPh, though still better than in PCAP. PCAPf achieves an average coverage of 87 percent with an average of 9 percent mispredicted shutdowns. The combined use of history and file descriptors is shown as PCAPfh in Fig. 11. The resulting average coverage is 86 percent with an average of 5 percent mispredicted shutdown. The energy saving in PCAPf and PCAPfh is also the same as in PCAP. On average, the mispredictions and energy savings did not change noticeably after adding file descriptors to the PCAPh. Only *mozilla* shows higher reductions in misses; thus adding file descriptor to PCAPh may be justified only for some workloads.

6.5.2 *Reuse and Storage Requirement of Prediction Tables*

More advanced predictors demand extended training which may not be provided during a single execution of the application. Fig. 12 compares PCAP and LT from Fig. 9 against PCAPa and LTa, which discard prediction tables after an application exits. Since PCAPa and LTa discard predictor information, they have to relearn predictions every time the application is executed. Training consumes a significant number of idle periods during which the backup predictor is responsible for making shutdown predictions to save energy.

The primary predictor (PCAP) with prediction table reuse is responsible for 71 percent of correct predictions, while the backup predictor provides an additional 16 percent of correct predictions, on average. Many studied applications do not have enough repetitive behavior to train the predictor and use its full potential during one execution. As a result, by discarding the trained prediction table, the primary predictor in PCAPa is responsible for only 15 percent of correct predictions, while the backup predictor provides 61 percent of correct predictions, on average. Similar behavior is observed in LT, where the primary predictor predicts 67 percent of hits and the backup predictor predicts 20 percent, on average. In LTa, on the other hand, the primary predictor only predicts 28 percent and the backup predictor 52 percent of hits, on average. We can also observe that mispredictions generally decrease in PCAPa since the primary predictors are making fewer predictions. The exceptions are *writer* and *impress*, where the backup predictor makes a significant amount of wrong predictions.

Fig. 13 and Fig. 14 show the prediction accuracy of LT and PCAP over consecutive executions of the applications. For clarity, only the results for three applications are shown. As can be seen, the prediction accuracy of PCAP gradually improves as more information is captured and stored in the prediction table. In contrast, the prediction accuracy of LT does not show significant improvement over multiple invocations. Interestingly, although the prediction accuracy of PCAP improves quickly, it does not converge to 100 percent. This is because the user behavior changes over time and there is a limit to the accuracy of any predictions based on the user's past behavior.

The higher energy savings is closely related to the prediction coverage of the primary predictor. Without prediction table reuse (PCAPa and LTa), most of the predictions are made by the backup timeout predictor; therefore, the overall energy savings are comparable to the simple TP. Thus, to achieve better energy savings than TP, it

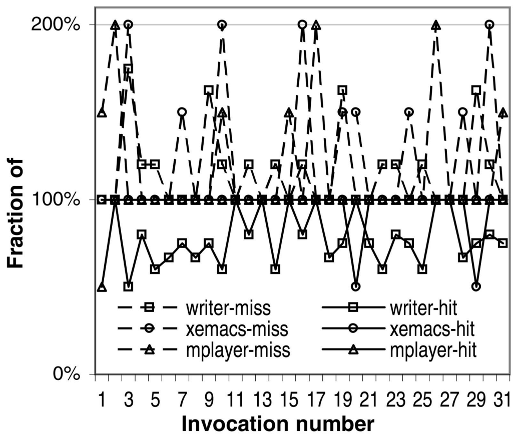

Fig. 13. Prediction table reuse over time for LT.

is important to perform application-based predictions. Implementation of sophisticated predictors without prediction table reuse does not provide significant gains to justify the complexity of the predictors.

Implementation of prediction table reuse saves the prediction table upon the application exit and reloads it when the new instance of the application starts executing. Table 4 shows the amount of information that is saved for each application. Each entry is encoded into a 4-byte word; therefore, even in the case of *mozilla*, which requires storing 139 entries in PCAPfh, the table consumes only 556 bytes. Other applications and predictors require even less storage and, therefore, the storage is not a problem in our experiments.

6.5.3 Low-Overhead Implementation

Fig. 15 shows the impact of low overhead design described in Section 4.4, which obtains the PCs only upon buffer cache misses. The experiment configuration is identical to the one in Fig. 11 except all predictors obtain PCs for the path during buffer cache misses. In this scenario, the PC is obtained much less frequently, resulting in a sparser path. A sparse path provides less information about the context and PCAP's mispredictions in *mozilla* and *impress* increased from 30 percent and 12 percent to 42 percent and 28 percent, respectively. Other applications are less affected, implying that the reduced information about the context of execution still provides enough information for correct prediction.

TABLE 4
Storage Requirements

Application	Number of entries			
	PCAP	PCAPh	PCAPf	PCAPfh
mozilla	72	99	129	139
writer	30	36	30	36
impress	34	44	44	47
xemacs	13	16	13	16
nedit	6	6	6	6
mplayer	24	24	26	26

The average number of mispredictions across the applications under PCAP increased from 12 percent to 16 percent. The number of correct shutdowns did not change and, as a result, the energy savings were mostly unaffected. However, higher mispredictions are not desirable and we apply techniques studied in Fig. 11 to reduce the number of mispredictions.

PCAPh shows similar impact as in Fig. 11, reducing the mispredictions in *mozilla* and *impress* to 21 percent and 7 percent, respectively. On average, PCAPh reduced the mispredictions to 8 percent across the applications, which is 2 percent higher than PCAPh in Fig. 11, which records the entire path. The average fraction of correctly predicted idle periods decreases to 86 percent, but no noticeable impact on energy savings is observed. PCAPf, which uses file descriptor, correctly predicts, on average, 87 percent idle periods with an average of 9 percent mispredictions. The combination of file descriptors and history in PCAPfh resulted in virtually identical behavior to PCAPfh in Fig. 11, which records all PCs. We thus conclude that adding file descriptors in addition to idle periods is unnecessary.

A reduction in the number of PC retrievals reduces the overhead in PCAP. The average hit ratio in the studied applications is 65 percent and, thus, this optimization will reduce the overhead of PC lookups by 65 percent. Furthermore, low mispredictions are still successfully achieved for most applications by including the idle period history in addition to the PC path. However, for *mozilla*, PCAPh with sparse paths increases mispredictions to 21 percent from 13 percent in PCAPh with full paths. Since a high number of mispredictions can affect the user experience, we use PCAPh with full paths in our Linux implementation study in the next section.

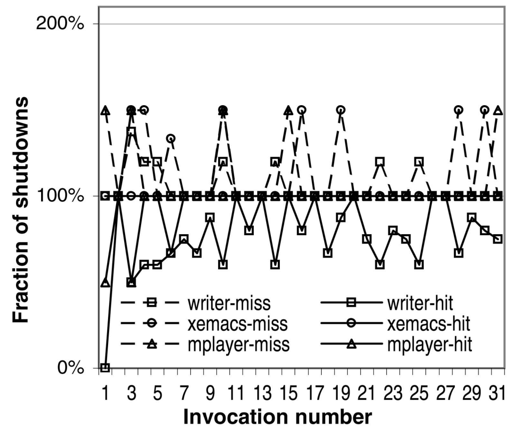

Fig. 14. Prediction table reuse over time for PCAP.

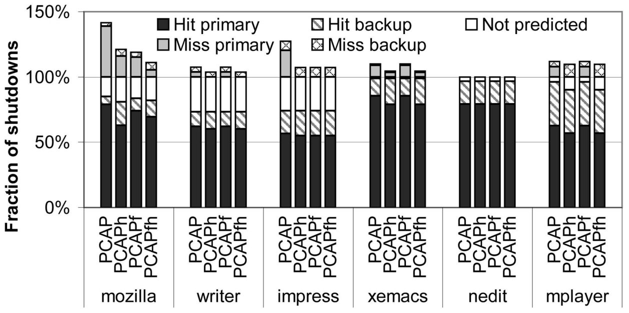

Fig. 15. Predictor optimizations using PCs obtained from buffer cache misses.

6.6 Simulation Summary

We summarize the various simulation results and highlight the benefits of the predictors and optimizations. We have observed that PCAP is much more accurate than LT while providing higher coverage and energy savings. Both predictors use sophisticated prediction techniques that require extensive training. Therefore, both predictors benefit from table reuse and the backup timeout predictor. We have observed that the small increase in storage required for history information in PCAP can significantly benefit some applications by reducing the number of mispredictions. Inclusion of file descriptors has a similar effect; however, significant storage requirements combined with limited improvement in prediction accuracy do not justify the implementation in real systems. Finally, we have shown that limiting the retrieval of PCs only during the buffer cache misses provides an alternative design, however, at a significant cost in accuracy for some applications.

From the above discussion, we conclude that the best performing implementation of PCAP in a real system should contain timeout backup, sliding-wait window, and table reuse, idle period history, and complete PC paths. File descriptors require high memory overhead therefore put pressure on resource allocation in the system, potentially limiting the available memory for buffer caching resulting in higher cache misses.

7 Implementation in Linux

In this section, we evaluate the performance of TP, LT, and PCAP running under Linux on a laptop computer. The timeout interval used in TP is set to 14 seconds. Both LT and PCAP use the sliding wait-window, table reuse, and backup timeout predictor optimizations, with the same 14-second interval for the timeout predictor as in TP. PCAP also includes the idle period history optimization to further reduce mispredictions. We first describe the implementation details of the predictors. We then present the energy savings and the energy-delay product which show PCAP as the best performer.

7.1 Implementation Details

PCAP has been implemented in Linux kernel version 2.4.20-30.9. The implementation presents three challenges: determination of the application program counter, integration of PCAP mechanisms into the unified buffer cache, and implementation of a global power manager to monitor predictions from all processes and schedule disk shutdowns.

7.1.1 Obtaining PCs of I/O Operations

There are multiple ways to obtain the PCs: library modification, system call interception, and kernel modification. In the first method, the modified library call can read the PC directly from the calling program's stack, therefore requiring the least amount of overhead. In the second method, interception of system calls happens at the user-kernel boundary, at which time the I/O library call may have invoked multiple levels of wrapper functions before finally invoking the system call. A time-consuming traversal of multiple library function stack frames may be necessary to retrieve the application's stack frame that invoked the library call. Finally, kernel modification is similar to system call interception, also requiring multiple stack frame traversals.

Library modification offers good performance, but requires extensive modifications to both the library code and kernel interfaces. To simplify the implementation, the kernel-only modification to obtain the PCs was selected. To determine the relevant PCs from within the kernel, we modified the `read` and `write` system calls. Upon each access to these calls, PCAP traverses the call stack to determine the relevant PCs. In our current implementation, this step is time-consuming for the following two reasons: First, multiple levels of stacks have to be traversed to determine the "signature PC." Second, each traversal involves repeatedly accessing user space from the kernel space. These user space accesses are expensive as they involve detailed checks to ensure proper access permissions, translation of user space addresses to kernel space, and, finally, copying data from user to kernel memory. In a production-quality implementation, the cost of PC determination can be reduced by reducing the number of times PCAP accesses the user space from the kernel space during stack traversals.

7.1.2 Interactions with the Buffer Cache

The buffer cache increases system performance by caching recently used files in memory. As a result, some of the I/O requests are served by the buffer cache without invoking actual disk I/Os. To accurately capture the path of I/O requests in the applications, PCAP monitors all I/O requests before they are filtered by the buffer cache. For every I/O operation, the signature PC is retrieved and the path so far is recorded in the process table. If the I/O request is a hit in the buffer cache, it is serviced by the cache without any further involvement of PCAP. When a cache miss and, consequently, an actual I/O occurs, PCAP performs a lookup in the prediction table. If the path is found in the process table, the prediction is made and communicated to the global predictor. The prediction table is updated with the new path only if the current miss is the first one after a long idle period.

The implementation of the unified buffer cache in the Linux kernel involves two separate cache structures—a page cache and a buffer cache. All file I/O operations to block devices go through the page cache which consists of pages, each storing several blocks of a file. Explicit block I/O operations also go through the buffer cache, which consists of buffers, each storing a single disk block. Thus, the blocks cached in the buffer cache are shared by the page cache and the sharing is implemented by the two caches actually storing references to the cached blocks. The two caches maintain separate meta-data, such as recency of accesses, but, because of the sharing of cached blocks, the management of the two caches is intertwined. For example, an access to a page entry in the page cache due to explicit I/O operations requires the corresponding block recency information to be updated in the block cache. Conversely, if a block entry in the buffer cache is accessed directly, e.g., by the flushing daemon, its access information also needs to be updated in the page cache. Modifying the cache management to support PCAP requires detailed examination of the cache management code to determine the appropriate location where PCAP can be integrated. Since all accesses to regular files (i.e., files that are not memory-mapped) are through two kernel functions, `generic_file_read` and `generic_file_write`, we modified these functions to call our predictor. However, at this point in the kernel, the data may be in the cache. Therefore, we also call our predictor if there is a cache miss. In summary, the predictor is informed of all I/O requests and of all cache misses, but only makes a prediction on a cache miss. A new state variable is also introduced in the process table to indicate the current prediction of a process. This prediction is then relayed on to the global predictor, as described in the next section.

The buffer cache also contains dirty data that needs to be flushed to the disk periodically. In the Linux operating system, this flushing is performed by a specialized daemon, `BdFlush`. By default, `BdFlush` flushes the data at a periodic interval of 30 seconds. The periodic flushing poses a challenge to saving energy as the disk will be spun-up by the `BdFlush` daemon even if the predictor has accurately predicted an idle period based on the application behavior. To reduce the interference between the disk accesses by `BdFlush` and shutdowns issued by PCAP, we increase the flushing interval to 10 minutes, based on a recent proposal in the Linux community [28] that suggests that extending the flushing interval to a few minutes can provide much higher energy savings with little impact on data integrity. For the applications we studied, 60 percent of the idle periods that are longer than the breakeven time are shorter than 3 minutes and 85 percent are shorter than 6 minutes. Thus, flushing of dirty data happens less frequently than most of the shutdowns. When the 10-minute flushing interval happens during a disk shutdown, `BdFlush` spins up the disk, performs the flush, and then immediately shuts down the disk as the global predictor still predicts the shutdown state.

7.1.3 Global Power Manager

The global predictor, Global Power Manager (GPM), is implemented as a kernel thread, `pcap`, which listens to predictions from individual processes. The communication between the processes is via two signals: `idle` and `busy`. An `idle` signal means that the process locally predicted shutdown and a `busy` signal means not shutdown. On receiving a signal from a process, GPM determines when to shut down the disk. The global shutdown determination is achieved as follows:

The main task of the GPM is to determine if all the processes are predicting shutdown. For each process that requires not shutting down, the GPM maintains a data structure which records the PID of the process and a time (T_e) before which the GPM expects the process to be not idle. T_e is calculated by adding the prespecified timeout (14 seconds) to the time when a busy signal is received. The data structure is kept sorted in order of decreasing T_e values. On receiving a `busy` signal, the GPM either creates a new entry for the process that sends the signal or updates the associated T_e value in the existing entry. The GPM will then sleep till the time is equal to the maximum value of T_e, i.e., $T_{e^{max}}$, unless a new signal arrives. If the GPM is able to complete its sleep uninterrupted, GPM removes all the entries as all of them have expired T_e values and the backup timeout predictor shuts down the disk as discussed in Section 4.3.

On receiving an `idle` signal from a process P, the GPM performs the following sequence: First, it determines if P has an associated T_e entry. If an entry exists, it is removed. Next, the GPM determines the current $T_{e^{max}}$ until a shutdown cannot be issued as other processes are expected to be busy. Note that if P was the process associated with the previous $T_{e^{max}}$ and, hence, just had its entry removed, the new $T_{e^{max}}$ will be the next largest value. The GPM then compares the value of $T_{e^{max}}$ with the current time. If the current time is after $T_{e^{max}}$, it implies no process is expected to be busy and a shutdown is issued. If the $T_{e^{max}}$ is in the future, GPM simply sleeps till then. The whole process repeats when new signals are received.

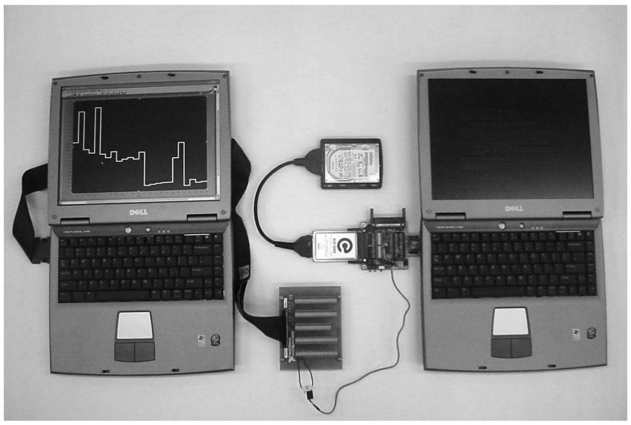
Fig. 16. The experimental setup used for measuring power. A data acquisition card is connected to the laptop on the left running Windows XP and the DAQ drivers and a 2.5-inch hard drive is connected to the laptop on the right running Linux.

As processes may exit without making a shutdown prediction, their associated data structure entries will persist and can cause a problem. To avoid this, on each busy prediction, the GPM examines the data structure and removes those whose T_e is less than the current time. In this way, extraneous entries will be pruned instead of consuming kernel resources.

Note that timeout filtering is also employed, i.e., even when a global shutdown is scheduled, GPM waits for the 1-second sliding wait-window before actually shutting the disk down. The shutdown of the disk is achieved via mechanisms similar to those of the `ioctl` interface for changing the disk state. No special step is needed to wake up the disk as that would happen automatically when the application accesses the disk. Moreover, before a shutdown is issued, the associated dirty blocks of the disk are flushed to reduce the likelihood of a later disk spin-up by the `Bdflush` daemon.

7.2 Results

7.2.1 Experimental Setup

The experiments were conducted using a setup of two Dell Inspiron Laptops with 1.5Ghz Intel Celeron Processor and 256MB of Memory, as shown in Fig. 16. The hard disk was connected via a PCMCIA measurement card to one of the laptops running Linux as described in Section 7.1. This drive is dedicated to the applications, i.e., the OS resides on the internal disk of the laptop. The data acquisition card (DAQ) was connected to the other laptop running Windows XP and the DAQ drivers. To measure the power consumed, a 0.1 ohms resistor was placed in series with the hard disk power supply and the voltage drop across this resistor was fed to DAQ. On the Windows machine, a LabView setup was programmed to sample the current measurements at 100Hz from DAQ and store it in a file.

A main issue in conducting the experiments was to subject the various energy saving techniques to the same benchmark executions so that the energy consumptions are comparable. To address this issue, we collected disk access traces of all the benchmarks similar to the trace of Section 6.1. Then, we designed a trace driver that takes as input a benchmark trace and replays the disk accesses exactly as they were issued by the benchmark. Hence, the trace driver emulates the benchmark executions in terms of disk accesses in real time.

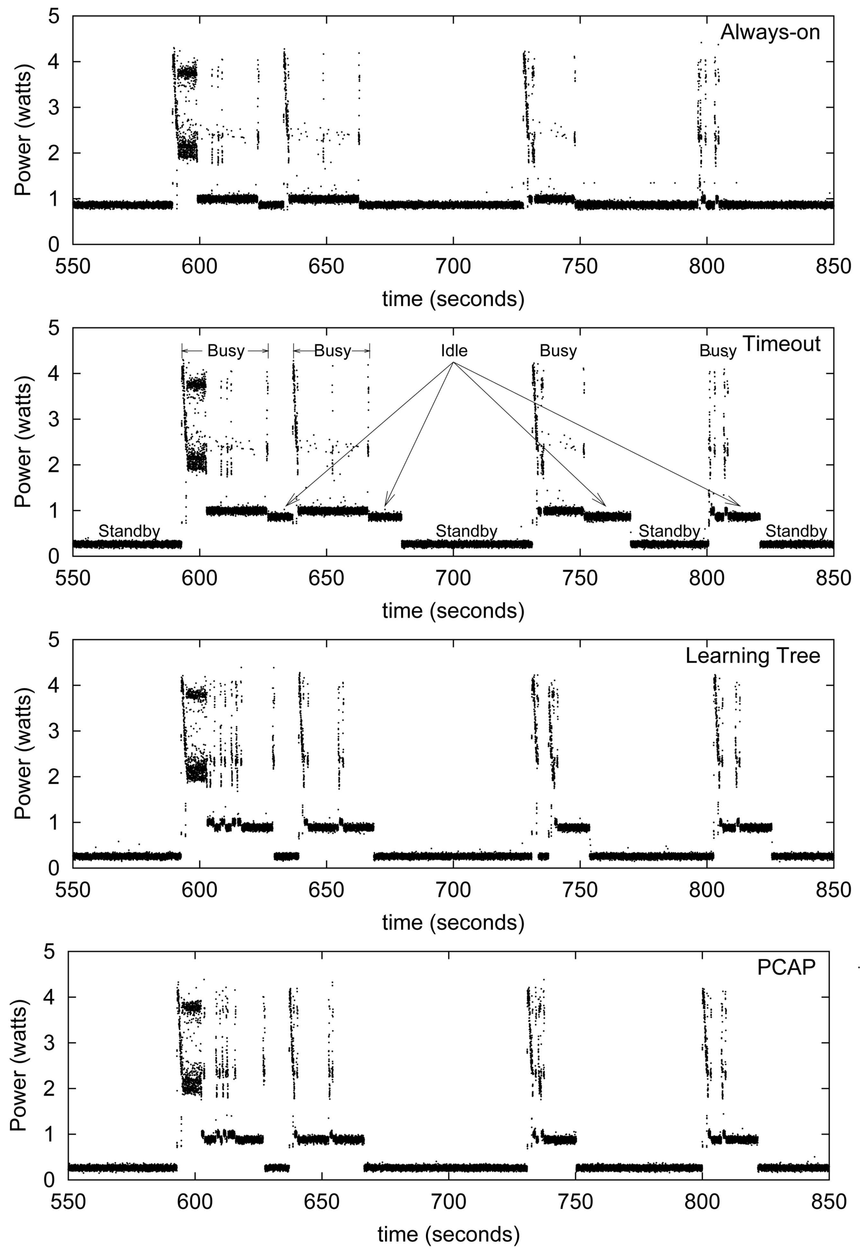

Fig. 17. Power consumption in a selected 300-second period during the *mozilla* execution under different energy saving schemes.

To calculate the energy, the data from DAQ was postprocessed. Since the voltage across the hard disk supply was found to be a constant of 5 Volts, the actual measured quantity, i.e., the current, was multiplied by 5 to obtain the instantaneous power. Finally, the energy consumption was determined using the Trapezium rule to estimate the area under the power curve, over the duration of the benchmark run.

7.2.2 Energy Savings

Fig. 17 shows the measured power for *mozilla* over a selected period of 300 seconds under four different scenarios: when no energy saving scheme was used (always-on) and when the timeout, Learning Tree, and PCAP schemes were employed. When no energy saving scheme was used, the disk remained switched on even when no accesses were made over long periods, e.g., before 595 seconds, 660 to 730 seconds, 750 to 790 seconds, and after 820 seconds.

For the simple timeout scheme, the disk is shut down for intervals longer than the timeout period, as can be seen in the timeout graph of Fig. 17. The figure separates the regions of busy, idle, and standby. During 550 and 592 seconds, the disk is in the low-power standby state. During 592 and 602 seconds, it is switching to the busy state. The disk serves requests during 592 and 628 seconds and is idle between 628 and 637 seconds. Because this idle period is shorter than the timeout, the disk does not enter the standby state. The disk becomes busy again during 637 and 669 seconds. It is idle between 669 and 680 seconds. This idle period is longer than the timeout value, so the disk enters the standby state at 680 seconds and remains in the standby state until 732 seconds.

Compared to the simple timeout scheme, the Learning Tree scheme was able to shut down the disk quickly by

TABLE 5
The Energy (in Joules) Consumed during the Execution of Various Benchmarks with No Power Saving Scheme, and with the Fixed Timeout, Learning Tree, and PCAP Shutdown Predictions

	mozilla	*writer*	*impress*	*xemacs*	*nedit*	*mplayer*
Always-on	27230.342	8638.318	7329.338	12240.032	15625.563	6800.412
TP	15601.938	4446.362	4193.776	7287.812	5496.387	4041.157
Learning Tree	13592.428	4278.704	3769.090	5895.368	5467.973	4042.728
PCAP	13511.080	4192.632	3679.962	5684.373	5315.792	3941.569

predicting idle periods. It also performed better by switching the disk off at 630 seconds, which was not detected by the timeout scheme, and improved the energy savings by shutting the disk down earlier, at 755 seconds instead of 14 seconds later at 769 seconds. However, at 735 seconds, Learning Tree mispredicted and shut down the disk, which was spun-up again by an access 7 seconds later. The PCAP graph shows that it performed largely similarly to the Learning Tree. However, it made fewer mispredictions. Notice that the disk was not shut down at time equal to 735 seconds as was the case for Learning Tree.

Table 5 shows the energy consumed during the execution of each application. All shutdown mechanisms save significant energy as compared to the base system in which the disk is never spun down. TP saves, on average, 47 percent of energy, LT is more agressive and increases the average energy savings to 51 percent. Fewer mispredictions in PCAP as compared to LT increase the average energy savings to 52 percent. The trends from the simulation result (Fig. 10) are present in Table 5. However, the energy saving numbers in the implementation also include the energy consumption due to kernel activities not related to the application.

For all benchmarks, the prediction techniques resulted in less than 3 percent increase in execution time. Some of the delays are present at the end of the application when the data is written to the disk and may not be noticeable by the user. Nevertheless, we include all delays in calculating the energy-delay product. The *energy-delay product* is the product of the execution time and disk energy consumption during the execution of the traces. Fig. 18 shows the energy-delay products of LT and PCAP normalized to that of TP. The figure shows that the overall impact of energy saving is beneficial for both techniques. Specifically, LT and PCAP have, on average, 8 percent and 10 percent, respectively, lower energy-delay product than that of TP.

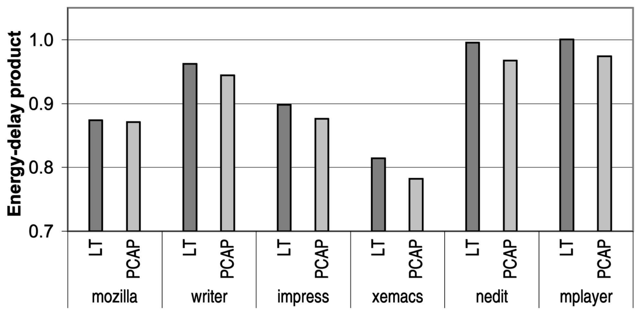

Fig. 18. Energy-delay product of LT and PCAP for the studied applications, normalized to that of TP.

Previously, Fig. 9 showed that both LT and PCAP have more shutdowns than TP. Every shutdown, either correctly predicted or mispredicted, requires a disk spin-up and, therefore, introduces a delay in program execution. Mispredicted shutdowns result in a double penalty on the energy-delay product since each mispredicted shutdown increases both the delay and the energy consumption. Fig. 18 shows that PCAP gives the best energy savings and lowest energy-delay product due to its ability to quickly and accurately predict the majority of the idle periods.

7.2.3 The Overhead of Obtaining PCs

To measure the overhead of obtaining PCs, we used a micro benchmark that repeatedly reads the same block of a file, which results in hits in the buffer cache. The time to service a hit in the buffer cache in the standard kernel is 3.3 microseconds. PCAP has to obtain the PC of the I/O operation, which took, on average, 3.2 microseconds. The total 6.5 microsecond overhead with the current simple implementation of PCAP is promising. These applications are not I/O intensive and, thus, the retrieval of PCs has limited impact on the overall execution time. Compared to LT, the extra overhead of obtaining PCs in PCAP is offset by its lower mispredictions, as shown in Section 6.3, which result in fewer mispredicted shutdowns and, thus, lower the delay on the execution time, As a result, the overall impact on the execution time for the two mechanisms are comparable, while PCAP achieves higher energy savings and consequently lower energy-delay product.

8 Conclusions and Future Directions

In this paper, we proposed the Program Counter Access Predictor, which dynamically learns the access patterns of the applications and predicts when the I/O device can be shut down in order to save energy. By using path-based correlation to observe access patterns, PCAP predicts future occurrences of long idle periods with high accuracy. We presented a detailed simulation study which showed that PCAP reduces the average mispredictions to 6 percent, which is lower than that of the timeout predictor (11 percent) and much lower than the mispredictions of Learning Tree (22 percent). We also showed the need for prediction table reuse to offset the delay of training (thus making predictions) in predictors more sophisticated than the timeout predictor. Our simulation results showed that table reuse reduces the training time and increases the prediction coverage of the primary predictors from, on average, 15 percent to 71 percent in PCAP and 28 percent to 67 percent in LT. Implemenation results showed that, while TP and LT save, on average, 47 percent and 51 percent of disk energy, respectively, PCAP is able to save, on average, 52 percent of disk energy. The reduction of energy-delay product corresponds closely to the energy savings: LT and PCAP achieve 8 percent and 10 percent lower energy-delay product, respectively, than TP.

PCAP can be further extended to handle multiple low power states of hard disks or other I/O devices. For example, the sliding wait-window can be optimized to put the disk into a lower power state immediately and only shut down after the wait-window elapses.

PCAP opens a new direction for the development of predictor-based techniques suitable for many other aspects of the operating system, such as file buffer management and I/O prefetching. PC-based techniques do not require any modification to an application and, yet, have the potential to obtain a program context similar to that provided by an annotated application. Thus, we expect PC-based predictions to perform as well as prediction schemes that rely on application hints.

Acknowledgments

The authors thank Le Cai for helping them set up the data acquisition card for energy measurement and the anonymous reviewers for their helpful comments. This work was supported in part by US National Science Foundation CAREER awards CCF-0238379 and CNS-0347466.

References

[1] J.-L. Baer and T.-F. Chen, "An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty," *Proc. 1991 ACM/IEEE Conf. Supercomputing,* Nov. 1991.

[2] N. Bellas, I. Hajj, and C. Polychronopoulos, "Using Dynamic Cache Management Techniques to Reduce Energy in a High-Performance Processor," *Proc. Int'l Symp. Low Power Electronics and Design,* Aug. 1999.

[3] 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.

[4] D. Bertozzi, L. Benini, and B. Ricco, "Power Aware Network Interface Management for Streaming Multimedia," *Proc. Wireless Comm. and Networking Conf.,* pp. 926-930, 2002.

[5] B. Black, B. Mueller, S. Postal, R. Rakvic, N. Utamaphethai, and J.P. Shen, "Load Execution Latency Reduction," *Proc. 12th Int'l Conf. Supercomputing,* July 1998.

[6] R.W. Carr and J.L. Hennessy, "WSCLOCK: A Simple and Effective Algorithm for Virtual Memory Management," *Proc. Eighth Symp. Operating Systems Principles,* Dec. 1981.

[7] E.V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," *Proc. Int'l Conf. Supercomputing,* June 2003.

[8] 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,* Oct. 2001.

[9] G.Z. Chrysos and J.S. Emer, "Memory Dependence Prediction Using Store Sets," *Proc. 25th Ann. Int'l Symp. Computer Architecture,* June 1998.

[10] 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.

[11] E.-Y. Chung, L. Benini, and G.D. Micheli, "Dynamic Power Management Using Adaptive Learning Tree," *Proc. Int'l Conf. Computer-Aided Design,* Nov. 1999.

[12] D. Colarelli and D. Grunwald, "Massive Arrays of Idle Disks for Storage Archives," *Proc. 15th High Performance Networking and Computing Conf.,* Nov. 2002.

[13] Dell Computer Corp., *Dell System 320SLi User's Guide,* June 1992.

[14] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive Disk Spin-Down Policies for Mobile Computers," *Proc. Second USENIX Symp. Mobile and Location-Independent Computing,* Apr. 1995.

[15] C.S. Ellis, "The Case for Higher-Level Power Management," *Proc. Workshop Hot Topics in Operating Systems,* Mar. 1999.

[16] K.I. Farkas, P. Chow, N.P. Jouppi, and Z. Vranesic, "Memory-System Design Considerations for Dynamically-Scheduled Processors," *Proc. 24th Ann. Int'l Symp. Computer Architecture,* June 1997.

[17] R.A. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness Is Not Sloth," *Proc. USENIX Winter Conf.,* Jan. 1995.

[18] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "DRPM: Dynamic Speed Control for Power Management in Server Class Disks," *Proc. Int'l Symp. Computer Architecture,* June 2003.

[19] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini, "Application Transformations for Energy and Performance-Aware Device Management," *Proc. 11th Int'l Conf. Parallel Architectures and Compilation Techniques,* Sept. 2002.

[20] Hewlett-Packard, "Kittyhawk Power Management Modes," internal document, Apr. 1993.

[21] C.-H. Hwang and A.C. 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.

[22] C. Im, H. Kim, and S. Ha, "Dynamic Voltage Scheduling Technique for Low-Power Multimedia Applications Using Buffers," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 34-39, 2001.

[23] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 197-202, 1998.

[24] Y. Jégou and O. Temam, "Speculative Prefetching," *Proc. Int'l Conf. Supercomputing,* July 1993.

[25] A.R. Karlin, M.S. Manasse, L.A. McGeoch, and S. Owicki, "Competitive Randomized Algorithms for Non-Uniform Problems," *Proc. First ACM-SIAM Symp. Discrete Algorithms,* Jan. 1990.

[26] A.-C. Lai and B. Falsafi, "Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction," *Proc. 27th Ann. Int'l Symp. Computer Architecture,* June 2000.

[27] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-Block Prediction and Dead-Block Correlating Prefetchers," *Proc. 28th Ann. Int'l Symp. Computer Architecture,* June 2001.

[28] J. Andrews, "Linux: Laptop Mode,, Prolonging Battery Life" http://kerneltrap.org/node/653, 2004.

[29] Y.-H. Lu, E.-Y. Chung, T. Simunic, L. Benini, and G.D. Micheli, "Quantitative Comparison of Power Management Algorithms," *Proc. Design Automation and Test in Europe,* Mar. 2000.

[30] Y.-H. Lu, G.D. Micheli, and L. Benini, "Requester-Aware Power Reduction," *Proc. Int'l Symp. System Synthesis,* Sept. 2000.

[31] J. Luo and N. Jha, "Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems," *Proc. Asia and South Pacific Conf. VLSI Design,* pp. 719-726, 2002.

[32] M.M.K. Martin, P.J. Harper, D.J. Sorin, M.D. Hill, and D.A. Wood, "Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared-Memory Multiprocessors," *Proc. 30th Ann. Int'l Symp. Computer Architecture,* June 2003.

[33] N. Megiddo and D.S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," *Proc. Second USENIX Conf. File and Storage Technologies,* Mar. 2003.

[34] R. Nair, "Dynamic Path-Based Branch Correlation," *Proc. 28th Ann. Int'l Symp. Microarchitecture,* Nov. 1995.

[35] R. Neugebauer and D. McAuley, "Energy Is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS," *Proc. Eighth Workshop Hot Topics in Operating Systems,* May 2001.

[36] 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,* pp. 59-64, 2001.

[37] Open Office, http://www.openoffice.org/, 2004.

[38] E. Pinheiro and R. Bianchini, "Energy Conservation Techniques for Disk Array-Based Servers," *Proc. 18th Int'l Conf. Supercomputing,* June 2004.

[39] S.S. Pinter and A. Yoaz, "Tango: A Hardware-Based Data Prefetching Technique for Superscalar Processors," *Proc. 29th Ann. ACM/IEEE Int'l Symp. Microarchitecture,* Dec. 1996.

[40] J. Pouwelse, K. Langendoen, and H. Sips, "Energy Priority Scheduling for Variable Voltage Processors," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 28-33, 2001.

[41] M.D. Powell, A. Agarwal, T.N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping," *Proc. 34th Ann. ACM/IEEE Int'l Symp. Microarchitecture,* Dec. 2001.

[42] Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes," *Proc. Design Automation Conf.,* June 1999.

[43] A. Ramachandran and M.F. Jacome, "Xtream-Fit: An Energy-Delay Efficient Data Memory Subsystem for Embedded Media Processing," *Proc. Design Automation Conf.,* pp. 137-142, 2003.

[44] G. Reinman and B. Calder, "Predictive Techniques for Aggressive Load Speculation," *Proc. 31st Ann. ACM/IEEE Int'l Symp. Microarchitecture,* Nov. 1998.

[45] M.T. Schmitz, B.M. Al-Hashimi, and P. Eles, "Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems," *Proc. Design Automation and Test in Europe Conf.,* pp. 514-521, 2002.

[46] T. Sherwood, S. Sair, and B. Calder, "Predictor-Directed Stream Buffers," *Proc. 33rd Ann. ACM/IEEE Int'l Symp. Microarchitecture,* Dec. 2000.

[47] T. Simunic, L. Benini, P. Glynn, and G.D. Micheli, "Dynamic Power Management for Portable Systems," *Proc. Int'l Conf. Mobile Computing and Networking,* Aug. 2000.

[48] J.E. Smith, "A Study of Branch Prediction Strategies," *Proc. Eighth Ann. Symp. Computer Architecture,* May 1981.

[49] M.B. Srivastava, A.P. Chandrakasan, and R.W. Brodersen, "Predictive System Shutdown and Other Architecture Techniques for Energy Efficient Programmable Computation," *IEEE Trans. VLSI Systems,* vol. 4, no. 1, pp. 42-55, Mar. 1996.

[50] A. Weisel, B. Beutel, and F. Bellosa, "Cooperative IO—A Novel IO Semantics for Energy-Aware Applications," *Operating Systems Design and Implementation,* pp. 117-129, 2002.

[51] A. Weissel, B. Beutel, and F. Bellosa, "Cooperative I/O—A Novel I/O Semantics for Energy-Aware Applications," *Proc. Fifth Symp. Operating System Design and Implementation,* Dec. 2002.

[52] C. Young, N. Gloy, and M.D. Smith, "A Comparative Analysis of Schemes for Correlated Branch Prediction," *Proc. 22nd Ann. Int'l Symp. Computer Architecture,* June 1995.

[53] H. Zeng, C.S. Ellis, A.R. Lebeck, and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 123-132, 2002.

[54] Q. Zhu, F.M. David, Y. Zhou, C.F. Devaraj, P. Cao, and Z. Li, "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," *Proc. 10th Int'l Symp. High-Performance Computer Architecture,* Feb. 2004.

[55] Q. Zhu, A. Shankar, and Y. Zhou, "PB-LRU: A Self-Tuning Power Aware Storage Cache Replacement Algorithm for Conserving Disk Energy," *Proc. 18th Int'l Conf. Supercomputing,* June 2004.

Chris Gniady received the BS degree in electrical and computer engineering from Purdue University in 1997 and the PhD degree in 2005 from the School of Electrical and Computer Engineering at Purdue University. He is an assistant professor of computer science at the University of Arizona. His research focuses on providing personalized computing by adopting software and hardware to the user's behavior. He uses instruction-based prediction and user behavior monitoring to optimize energy and performance in computer systems. He is a member of the IEEE.

Ali R. Butt received the BSc (Hons) degree in electrical engineering from the University of Engineering and Technology Lahore, Pakistan, in 2000. He is currently a PhD candidate in computer engineering at Purdue University, where he also served as the president of the Electrical and Computer Engineering Graduate Student Association for 2003 and 2004. His research interests lie broadly in distributed systems and operating systems. In particular, he has explored distributed resource sharing systems spanning multiple administrative domains, applications of peer-to-peer overlay networking to resource discovery and self-organization, and techniques for ensuring fairness in sharing of such resources. His research in operating systems focuses on techniques for improving the efficiency of modern file systems via innovative buffer cache management. He is a member of USENIX, the ACM, and the IEEE.

Y. Charlie Hu received the MS and MPhil degrees from Yale University in 1992 and the PhD degree in computer science from Harvard University in 1997. He is an assistant professor of electrical and computer engineering and computer science at Purdue University. From 1997 to 2001, he was a research scientist at Rice University. His research interests include operating systems, distributed systems, networking, and parallel computing. He has published more than 70 papers in these areas. He received the Honda Initiation Grant Award in 2002 and the US National Science Foundation CAREER Award in 2003. He served as a TPC vice chair for the 2004 International Conference on Parallel Processing, and a cofounder and TPC cochair for the International Workshop on Mobile Peer-to-Peer Computing. He is a member of USENIX, the ACM, and the IEEE.

Yung-Hsiang Lu (S'90-M'03) received the PhD degree in electrical engineering from Stanford University, California, 2002. He is an assistant professor in the School of Electrical and Computer Engineering and, by courtesy, the Department of Computer Science at Purdue University, West Lafayette, Indiana. In 2004, he received the US National Science Foundation Career Award for studying energy management by operating systems. His research focuses on energy management for computer systems, mobile robots, sensor networks, and image processing. He is a member of the IEEE.