Program Counter Based Techniques for Dynamic Power Management

Chris Gniady, Y. Charlie Hu, and Yung-Hsiang Lu
School of Electrical and Computer Engineering, Purdue University
{gniady,ychu,yunglu}@purdue.edu

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. The paper 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 better energy savings while incurring low mispredictions.

1. Introduction

Reducing energy consumption has become one of the most important challenges in designing future computing systems. While Moore's Law provides steady reduction in power consumption per operation, increasing demand for higher performance, versatile functionalities, and better user interfaces have 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 [16] 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 later needed, it is turned on. This is called dynamic power management. Unfortunately, there are overheads to shut down and turn 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 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 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 the 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 PC is repeated in the same context and was previously followed by a long idle period, then the predictor 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 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 unavail-

In Proceedings of the 10th International Symposium on High-Performance Computer Architecture (HPCA-10), Madrid, Spain, February 14-18, 2004.

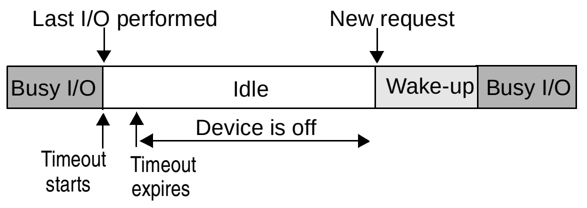

Figure 1. Anatomy of an idle period.

able in any of the existing methods. Because of the precise information, our method is able to attain better energy savings with very few mispredictions.

The rest of the paper is organized as follows. Section 2 describes current energy saving techniques. In Section 3, we present the motivation for path-based prediction and the design of PCAP. Section 4 presents optimizations for reducing mispredictions and training time. Section 5 describes the global predictor for the multiprocess environment. Section 6 presents simulation results and finally, Section 7 concludes the paper.

2. Background: Current predictors

Many computers use power management to reduce energy consumption. Since early 1990s, manufacturers have been recommending spinning down hard disks after some period of idleness [4, 9]. The simple timeout mechanism has gained wide popularity and is currently implemented in many operating systems. Figure 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 Figure 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. [11] 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 interrupt the user, the timeout 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, more sophisticated shutdown techniques were proposed. Dynamic predictors were proposed that completely eliminate the timeout interval [3, 10, 20]. These methods predicted the length of an idle period before a device became idle. In [20], Srivastava et al. suggested that the length of an idle period could be predicted by the length of the previous busy period. A long idle period often followed a short busy period. Chung et al. [3] considered the pattern of sequences of idle periods and constructed a search tree. When an idle period occurred, the power manager would find a path that best matched the sequence that led to the current idle period and predicted the length of the current idle period. Hwang et al. [10] 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 timeout [5, 7]. Both methods used feedback to enlarge or to reduce the timeout based on whether the previous prediction was correct. If it was correct, the timeout was reduced; otherwise, it was enlarged.

Stochastic modeling [1, 2, 17, 18] was also used to model the trace behavior and predicted the idle period based on the model parameters. In these approaches, I/O requests were considered as a stochastic process. Benini et al. [1] 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. [2] extended the method and considered non-stationary accesses. Their method pre-computed the optimal solutions for several I/O probabilities. At runtime, the power manager estimated the current probability and interpolated from the pre-computed solutions. Qiu et al. [17] used continuous-time Markov models and event-triggering so the power manager would not have to periodically re-evaluate whether to shut down a device. Simunic et al. [18] 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 predictors is presented in [13] 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 off-line preprocessing and are more difficult to implement, and problems may arise if the workload changes [2]. Application controlled power management [6, 8, 14, 21] has much better potential for reducing energy consumption. However, the technique places an additional 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 predictor. In this paper we adopt sophisticated branch prediction techniques for energy management.

2.1. Branch prediction based techniques

Dynamic predictors are based on the premise that a history of events is likely to repeat in the future due to repetitive behavior of the applications [19]. Learning Tree [3] is the first attempt to adapt branch prediction techniques for energy management. Figure 2 shows an example of some repetitive behavior of idle periods. Learning Tree discretizes the idle periods and uses the patterns to make prediction. In Figure 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.

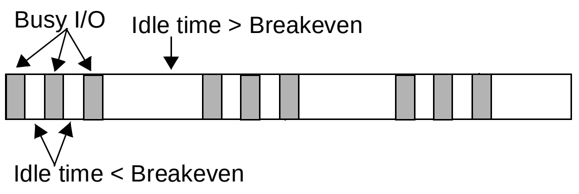

Figure 2. Repetitive behavior of I/O accesses.

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 instructions within an application and the immediate 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 predict future idle periods based on previous experiences. Thus PCAP differs from existing methods that lack the detailed 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.

To address these problems, PCAP records a *path* which is 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 [15] and was later successfully used in predicting cache block eviction [12].

Figure 3 shows an example of I/O operations made by an application. The leftmost column is the program counters that initiate I/O operations. The middle column

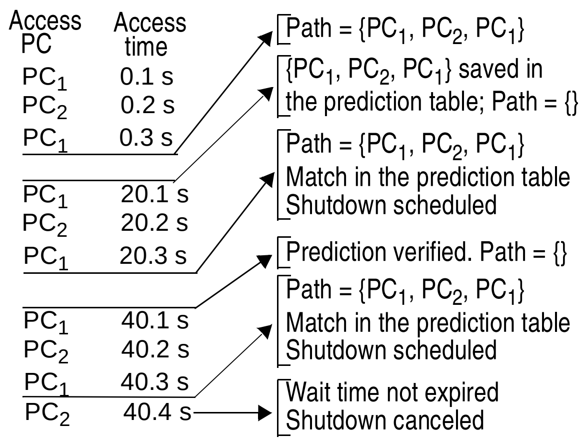
Access PC
Access time
PC1 0.1 s
PC2 0.2 s
PC1 0.3 s
PC1 20.1 s
PC2 20.2 s
PC1 20.3 s
PC1 40.1 s
PC2 40.2 s
PC1 40.3 s
PC2 40.4 s
Path = {PC1, PC2, PC1}
{PC1, PC2, PC1} saved in the prediction table; Path = {}
Path = {PC1, PC2, PC1} Match in the prediction table Shutdown scheduled
Prediction verified. Path = {}
Path = {PC1, PC2, PC1} Match in the prediction table Shutdown scheduled
Wait time not expired Shutdown canceled

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

shows the time when an I/O operation occurs. The right column shows 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 $\{PC_1, PC_2, PC_1\}$. 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 sequences 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 $\{PC_1, PC_2, PC_1\}$ triggers the prediction of an idle period and the disk is shut down. The example also shows the third sequence of $\{PC_1, PC_2, PC_1\}$ that is immediately followed by PC_2. In this case, the misprediction will occur if there is no additional information present.

3.2. Basic design

So far we have discussed predicting idle periods based on a path of I/O triggering PCs followed 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 [12] for predicting cache accesses. Such encoded path results in a 4 byte variable, called a *signature* in the rest of the paper. For example, a path $\{PC_1, PC_2, PC_1\}$ from Figure 3 is encoded as

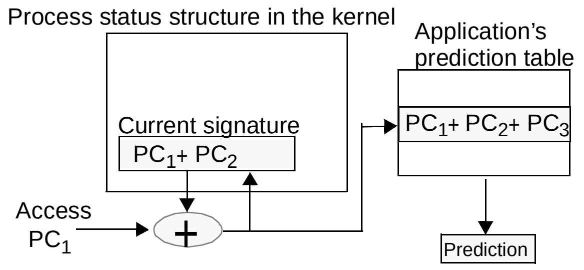
Process status structure in the kernel
Application's prediction table
Current signature
PC1+ PC2
PC1+ PC2+ PC3
Access PC1
+
Prediction

Figure 4. Implementation of PCAP.

$PC_1+PC_2+PC_1$. The encoding minimizes the storage requirements of PCAP and provides a quick comparison between current signature and the signature in the prediction table. Such encoding introduces the possibility of two different paths resulting in the same signature. For example, path $\{PC_1, PC_2, PC_1\}$ is different from path $\{PC_1, PC_1, PC_2\}$, but it will result in the same signature. In our experiments, this signature aliasing did not occur. Therefore we do not explore alternative encodings.

Figure 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 breakeven, the current signature variable in the current process is overwritten by 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, PCAP uses the signature to lookup the prediction in the prediction table. If a signature match is found between the current signature and a 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 signature in the prediction table. After PCAP learns the new signature, it will use it for the future predictions.

3.2.1. Obtaining PCs of I/O operations PCAP needs the PC of the function call from the applications that invoked and I/O operation. There are multiple ways to obtain the PC: library modifications, system call interceptions, and kernel modifications. 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 the system calls happens at the user-kernel boundary, at which time the I/O library call may have invoked multiple levels of functions which finally invoked the system call. A time consuming traversal of multiple library function stack frames may be necessary to arrive at the application's stack frame that invoked the library call. Finally, the kernel modifications are similar to the system call interceptions, also requiring multiple stack frame traversals. Library modifications are preferable since the PC can be obtained as soon as the application starts executing the library code and stack frame traversal is needed.

3.2.2. Runtime Overhead of PCAP During each I/O operation, PCAP has to obtain the PC, calculate the signature, and perform the predictor table lookup. However, these steps can be implemented very efficiently. To obtain the PC and calculate the signature requires about four memory accesses when the I/O system library is modified to obtain PCs. The predictor lookup consists of a hash table access and the comparison of signatures. These overheads are insignificant with respect to time and energy consumption as compared to thousands of instructions required to process an I/O operation.

4. PCAP optimizations

PCAP is able to retrieve and make use of program context and as a result, it can achieve high number of predicted shutdowns 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 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 Figure 3 shows the occurrence of subpath aliasing. The path $\{PC_1, PC_2, PC_1\}$ is the subpath of $\{PC_1, PC_2, PC_1, PC_2\}$. 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 can be obtained from an Internet browser where some pages require loading additional libraries (additional I/Os) to decode the multimedia context and some do not.

4.1.1. Sliding wait-window To reduce the mispredictions due to subpath aliasing, PCAP uses a *sliding wait-window* filter before shutting down the disk. In Figure 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. History and file descriptors The wait-window is unable to eliminate all mispredictions caused by subpath aliasing. As a solution, we provide PCAP with additional information about the context which will help PCAP in distinguishing different paths and reducing subpath aliasing. We propose two sources of additional information: history of idle periods and file descriptor of the I/O operation. These sources are orthogonal and can be implemented concurrently to further improve the accuracy of PCAP.

History based prediction is drawn from the wealth of optimizations proposed for branch predictors. We incorporate 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 run time. Paths 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.

Inclusion of file descriptors into the predictor table entries is motivated by research in [12], where the authors use the address of the cache block to aid the predictor in differentiating cache blocks that exhibit subpath aliasing. The direct adoption would be to use the location of accessed files on the disk. However, inclusion of file locations makes the predictor table size dependent on the I/O footprint of the application, and we would face the same problem of exploding predictor table size as occurs in [12]. 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, on the other hand, show less variability and provide related context, because file descriptor are often assigned based on some user behavior.

4.2. Reusing prediction tables

Path-based prediction requires extensive learning to populate the prediction table. To reduce the delay in learning, we propose to reuse the prediction tables across multiple executions of the same application. While PCAP uses learning 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. The prediction table is loaded when the application starts again and reads the initialization file.

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. The old entries can be replaced when the new behavior is detected. A simple LRU mechanism would be sufficient in removing old unused entries.

4.3. Backup predictor

Prediction table reuse significantly reduces predictor training, but application or user behavior may change over time and thus it is not possible to eliminate future training. During training, on a particular signature, PCAP does not make a shutdown prediction, and the disk will remain spinning for the entire idle period used for training. To reduce the impact of training on energy savings, we use the simple timeout predictor as the backup predictor for PCAP. When PCAP is unable to match a signature, the backup timeout predictor shuts down the disk after the timer expires. This is the only time when the timeout predictor overrides the no-idle prediction from PCAP and shuts down the disk.

5. Global prediction

So far we have discussed PCAP implementation and optimizations on a per application basis. In real systems, many processes are running concurrently and some of them may be from a single application. To generate shutdown prediction, a system-wide prediction is needed that will take into account multiple processes running concurrently. Figure 5 presents the Global Shutdown Predictor that generates global shutdown prediction by considering the input from all processes. Each process has its own private PCAP which generates local predictions as shown in Figure 4. The Global Shutdown Predictor predicts shutdown only when the PCAP for every process in the system predicts shutdown.

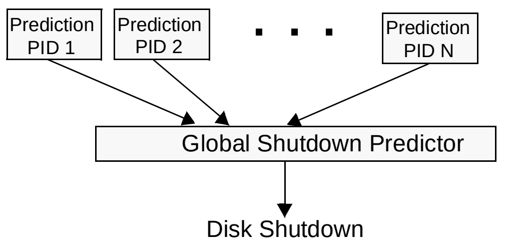

Figure 5. Global predictor.

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 I/O operation that wakes up the disk. When an idle period occurs, the prediction will be generated by every application. 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 Figure 5 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 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 PCAP from process 2 makes the prediction. We can observe that PCAP from currently running process will make the last prediction and no synchronization is necessary between waiting processes at this time.

6. Results

To evaluate the performance of PCAP and compare it to previously proposed predictors we use a trace simulator. A detailed trace of the applications was obtained by modifying the `strace` Linux utility. The modified `Strace` reads traced processs 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 1 shows five applications used by a user during the trace collection. 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

Appl.	Num. of executions	Num. of idle periods		Total I/Os
		Global	Local	
mozilla	49	365	1001	90843
writer	33	112	358	133016
impress	19	87	234	220455
xemacs	37	94	103	79720
nedit	29	29	29	6663
mplayer	31	51	111	512433

Table 1. **Applications and execution details.**

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 single process. Writer is a word processor from the Open Office suite 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 1 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.

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

To take into account of the effects of disk caching in an operating system, we have 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 256 Kbytes. We use the LRU mechanism for cache replacement and the default timer of 30 seconds between cache flushes of dirty data. Since the studied applications did not generate large amount of data the impact of dirty data flushes was limited. The elongation of default timer and optimizations of dirty data flushes are being currently evaluated in the Linux community. These optimizations will further benefit the power management.

State	**Power**
Busy power	2.2 W
Idle power	0.95 W
Standby power	0.13 W
Spin-up energy	4.4 J
Shutdown energy	0.36J
State Transition	**Delay**
Spin-up time	1.6 sec.
Shutdown time	0.67 sec.
Breakeven time	5.43 sec.

Table 2. **The states and state transitions of the simulated disk.**

Energy consumption and savings are calculated based on the amount of time the applications spend in a particular state and the corresponding power consumption as listed in Table 2. These parameters correspond to Fujitsu MHF 20043 AT disk drive [13].

We start by evaluating the ability of the predictors to predict shutdowns in Section 6.1 and Section 6.2. In Section 6.3, we evaluate energy savings of various predictors. In Section 6.4, we compare the effectiveness of different optimizations of PCAP for reducing mispredictions and learning time.

6.1. Local prediction accuracy

In this section, we compare the accuracy and the ability of predictors to predict hard disk shutdowns. In Figure 6 we compare the timeout predictor (TP), the Learning Tree (LT) predictor, and PCAP. TP uses a 10-second timer and after the timer expires it shuts down the disk. The 10-second interval results in low mispredictions and good energy savings in our applications. Lower timer values would increase mispredictions significantly and much higher timeout would reduce the energy savings considerably. The 10-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 only predicts shutdowns. The backup timeout predictor and the sliding wait-window mechanism are included in both LT and PCAP, allowing a direct comparison. We have used one-second wait-window since it filters mispredictions in most common cases.

Figure 6 presents the fractions of shutdowns normalized to the number of idle periods that are long

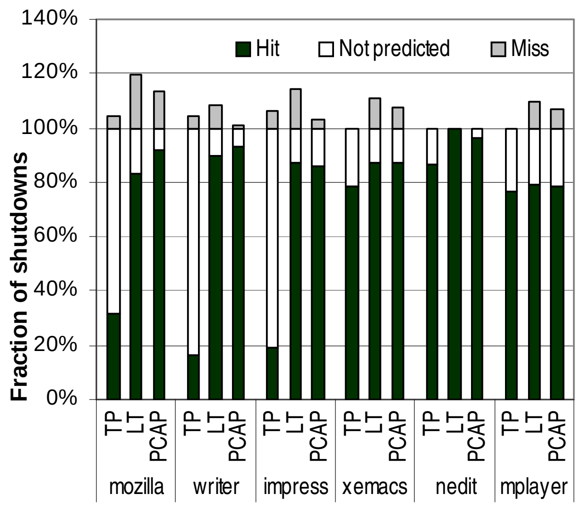

Figure 6. Local shutdown predictor

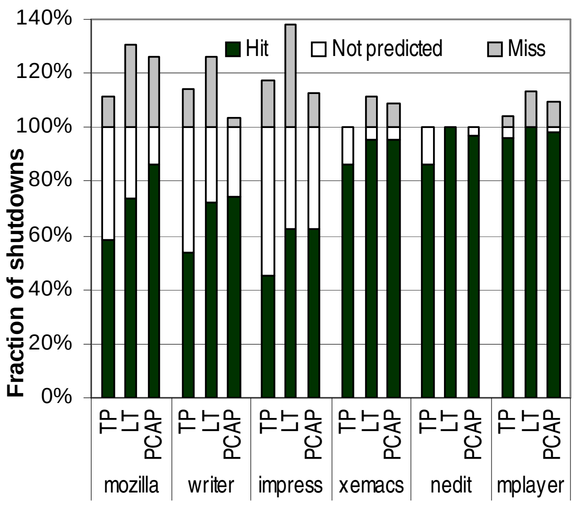

Figure 7. Global shutdown predictor

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 presents missed opportunity to shut the disk down. The fraction of "Miss" corresponds to the additional shutdowns that were introduced due to misprediction. The additional shutdowns occurred during the idle periods that were shorter than the breakeven time and therefore are not part of idle periods shown in Table 1. However, we normalized the misprediction to the number of idle periods for direct comparison in the figures.

The complete system-wide predictor contains one global and many local predictors, as shown in Figure 5. Figure 6 presents results for the local predictor by calculating the total number of misses and hits from each process and normalizing them to the total number of idle periods for the processes in the application. High accuracy in the local predictors will result in improved accuracy for the global predictor. TP has the lowest number of predictions, 52% on average, and as a result it has the lowest number of mispredictions, 3% on average. 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 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 10% across the applications. LT is able to correctly predict 88% of local shutdowns. To maximize energy savings and minimize mispredictions we have used a history length of eight in LT. Longer history lengths does not improve accuracy. Shorter history may result in more hits, but misprediction may also increase.

PCAP achieves the highest average coverage by correctly predicting 89% of the local shutdown intervals. Here, the *coverage* is defined as correctly predicted shutdowns as a percentage of all such opportunities. 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. Since PCAP has more sophisticated learning mechanisms, it requires more training than the predictors that do not observe the application context. PCAP also improves the prediction accuracy, compared to LT, with only 5% mispredicted shutdowns. Compared to TP, PCAP has 37% more coverage and only 2% more mispredictions. The mispredictions in PCAP can be significantly reduced by providing more context as shown in Section 6.4.

6.2. Global prediction accuracy

The final shutdown prediction is made by the global predictor, which is based on the collection of local predictions. Therefore, in the remaining sections we will only present global prediction results. Figure 7 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 the predictors should attempt to shut down the disk. Figure 7 follows the trends of Figure 6 except for the following differences. First, TP achieves much higher percentage of hits than in Figure 6. This can be explained by the

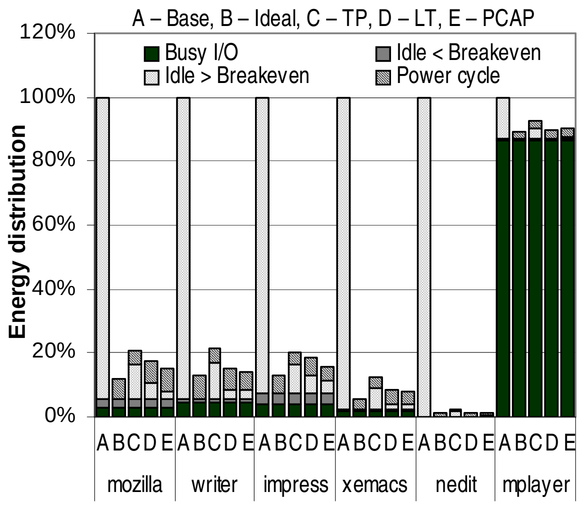

Figure 8. Energy distribution.

lower number of global idle periods the predictions are normalized against, as explained before. Second, LT and PCAP achieve lower percentage of hits than in Figure 6. This is caused by mixed TP (backup) and LT or PCAP as local predictors. Since TP requires a much large timeout period (10 seconds) before predicting a shutdown, while LT and PCAP can make predictions immediately. If any local predictor is using TP, the global prediction has to wait for 10 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 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 71% of idle periods, on average, while incurring only 8% of mispredicted shutdowns. LT is more aggressive with an average of 84% correct shutdowns, but also incurs an average of 20% mispredicted shutdowns. PCAP again predicts much better than LT, correctly shutting down the disk during 86% of the idle periods, on average, while incurring only 10% of mispredicted shutdowns. The overall trends across applications remain unchanged from Figure 6.

6.3. 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. Figure 8 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” section for 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 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 83% of energy is consumed during the idle I/O state, and 82% of energy is from the intervals longer than the breakeven time. The exception is mplayer which requires 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 the Figure 8 corresponds to the amount of time it took to empty the 8 MB buffer at the end of the movie. The idle time in the other applications depends on the user interaction. 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 like dictionaries or graphic filters that require more I/O time.

The ideal predictor in Figure 8 saves all energy that comes from the idle periods that are longer than breakeven time. The energy required to turn off and on the disk 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% of energy in the applications.

Simple TP performs well, on average saving 72% of energy in the applications which is 6% away from the ideal predictor. The energy savings of TP can be increased at the cost of higher mispredicted shutdowns by setting the timeout to be the breakeven time [11]. In this case, TP with timeout of 5.43 seconds eliminates on average 74% of energy, however the global mispredictions increase to 12% as a result of shorter timeout.

LT is more aggressive in making predictions and saves on average 75% of energy. PCAP predictor saves

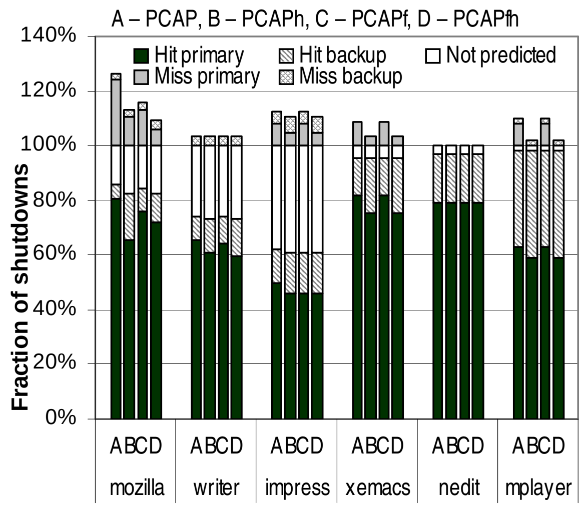

Figure 9. Predictor optimizations

on average 76% of energy which is only 2% 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 also can affect disk reliability and irritate the user who has to wait for the disk to spin up.

Timeout plays a role in all predictors since LT and PCAP use timeout prediction as a backup during training intervals. 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 application for TP, making it even more difficult to select a value that will benefit a wide range of applications. LT and PCAP energy savings are not affected by the timeout value, since most predictions are handled by the primary predictors in LT and PCAP. Aggressive timeout values do not benefit PCAP and a longer timeout is preferable, because it eliminates mispredictions due to short timeout.

6.4. Optimizations

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

6.4.1. History and file descriptors Prediction accuracy is improved by providing the predictor with additional information about the context of execution. We present PCAP in Figure 9 with the addition of idle period history (PCAPh), file descriptor (PCAPf), and combination of history and file descriptor (PCAPfh), respectively. Figure 9 presents results for the global predictor, and the base PCAP from Figure 7 is included for comparison. Each misprediction and hit section of the bar was split into two sections to show the contribution from the primary and backup predictor. Since there were multiple processes 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 Figure 7, achieving high coverage of 85% at a relatively low cost of only 10% 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 remains at 85%, but the additional context provided by history results in drop in the mispredictions to an average of 5%. At this point PCAPh achieves higher coverage and fewer mispredictions than TP or LT. The impact of using history on energy savings is limited and results in well under 1% average change. As a result PCAPh is still able to save 76% of energy at a cost of only 5% mispredicted shutdowns.

Mozilla is the most difficult to predict, however PCAPh manages to reduce the misprediction rate to 13% as compared to 26% in PCAP. The 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. The additional complexity that history introduces is well justified in case of mozilla, and other applications are seeing limited benefit as well.

Addition of file descriptors to the path of PCs (PCAPf) also improves the predictability. 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 85% with an average of 9% mispredicted shutdowns. The combined use of history and file descriptors is shown as PCAPfh in Figure 9. The resulting average coverage is 84% with an average of 5% mispredicted shutdown. The energy saving in PCAPf

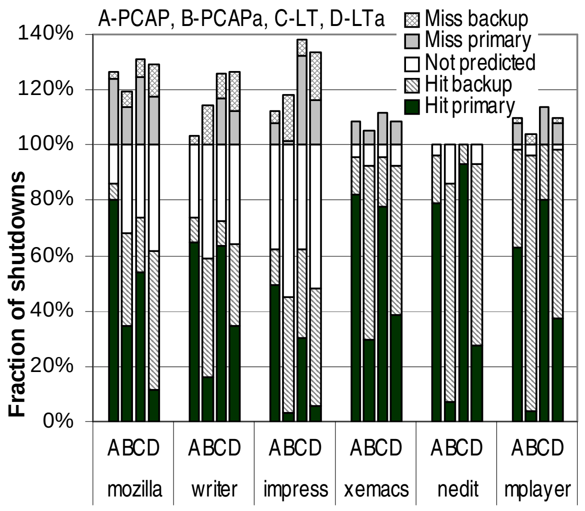

Figure 10. Predictor table reuse

and PCAPfh is also the same as in PCAP. On average, the mispredictions and energy savings did not change after adding file descriptors to the PCAPh. Only mozilla shows higher reductions in misses, therefore adding file descriptor to PCAPh may be justified only for some workloads.

6.4.2. Reuse and storage of prediction tables More sophisticated predictors demand extended training that a single execution of the application may not provide. Figure 10 shows PCAP and LT from Figure 7 and compares them against PCAPa and LTa which discard predictor tables after an application exits. Since PCAPa and LTa discard predictor information, they have to relearn prediction signatures 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 70% of correct predictions, while the backup predictor provides additional 15% of correct predictions, on average. Studied applications mostly do not have enough repetitive behavior to train the predictor and use its full potential during one execution. As a result, by discarding trained predictor table, the primary predictor in PCAPa is responsible for only 16% of correct predictions while the backup predictor provides 59% of correct predictions, on average. Similar behavior is observed in LT where the primary predictor predicts 66% of hits and the backup predictor 18%, on average. In LTa, on the other hand, the primary predictor only predicts 26% and the backup predictor 50% of hits, on average. We can also observe that mispredictions generally decrease in PCAPa since primary predictors are making fewer predictions. The exceptions are writer and impress where the backup predictor makes significant amount of wrong predictions.

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

Table 3. Storage requirements.

The higher energy savings relate to the prediction coverage of the primary predictor. Without prediction table reuse (PCAPa and LTa), most of 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 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 3 shows the amount of information that is saved for each application. Each entry is encoded into a 4-byte word, therefore even in case of mozilla which requires to store 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. Longer running predictors and changing user behavior can result in many more signatures. In this case, some storage limit can be imposed and an LRU replacement of old signatures can be used.

7. Conclusion

In this paper we have proposed 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 present implementation of PCAP that reduces average mispredictions to 5% which is much lower than the mispredictions in Learning Tree and even lower than in Timeout Predictor. We have also shown the need for prediction table reuse to offset the overhead of training in predictors more sophisticated than the timeout based predictors. Our experimental results show that table reuse reduces the training time, resulting in on average fourfold increase in PCAPs coverage and more than doubles the coverage of LT. Overall, PCAP is able to save on average 76% of the total energy consumed by I/Os, only 2% away from a perfect predictor savings.

PCAP can be further extended to handle multiple low power states of hard disks. 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 program context as accurate as one provided by an annotated application. Thus we expect PC-based predictions to perform as well as prediction schemes that rely on application hints.

References

[1] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, June 1999.

[2] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. D. Micheli. Dynamic Power Management for Nonstationary Service Requests. *IEEE Trans. on Computers*, 51(11):1345–1361, November 2002.

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

[4] Dell Computer Corp. Dell System 320SLi User's Guide. June 1992.

[5] F. Douglis, P. Krishnan, and B. Bershad. Adaptive Disk Spin-down Policies for Mobile Computers. In *Proceedings 2nd USENIX Symp. on Mobile and Location-Independent Computing*, pages 381–413, 1995.

[6] C. S. Ellis. The Case for Higher-Level Power Management. In *Workshop on Hot Topics in Operating Systems*, pages 162–167, Rio Rico, AZ, USA, March 1999.

[7] R. A. Golding, P. B. II, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is Not Sloth. In *Proceedings of the USENIX Winter Conference*, pages 201–212, 1995.

[8] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Application transformations for energy and performance-aware device management. In *Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques*, September 2002.

[9] Hewlett-Packard. Kittyhawk power management modes. *Internal document*, April 1993.

[10] C.-H. Hwang and A. C. Wu. A Predictive System Shutdown Method for Energy Saving of Event Driven Computation. *ACM Trans. on Design Automation of Electronic Systems*, 5(2):226–241, April 2000.

[11] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive Randomized Algorithms for Non-Uniform Problems. In *Symposium on Discrete Algorithms*, pages 301–309, 1990.

[12] A.-C. Lai and B. Falsafi. Selective, accurate, and timely self-invalidation using last-touch prediction. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

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

[14] Y.-H. Lu, G. D. Micheli, and L. Benini. Requester-aware power reduction. In *Proceedings of the International Symposium on System Synthesis*, pages 18–24, 2000.

[15] R. Nair. Dynamic path-based branch correlation. In *Proceedings of the 28th annual international symposium on Microarchitecture*, pages 15–23, November 1995.

[16] R. Neugebauer and D. McAuley. Energy is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS. In *Workshop on Hot Topics in Operating Systems*, pages 59–64, 2001.

[17] Q. Qiu and M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In *Proceedings of the Design Automation Conference*, pages 555–561, New Orleans, LA, USA, June 1999.

[18] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Dynamic Power Management for Portable Systems. In *Proceedings of the International Conference on Mobile Computing and Networking*, pages 11–19, 2000.

[19] J. E. Smith. A Study of Branch Prediction Strategies. In *Proceedings of the 8th Annual Symposium on Computer Architecture*, pages 135–148, 1981.

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

[21] A. Weissel, B. Beutel, and F. Bellosa. Cooperative I/O—a novel I/O semantics for energy-aware applications. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation*, December 2002.