Self-Learning Hard Disk Power Management for Mobile Devices

Andreas Weissel
University of Erlangen-Nuremberg
Distributed Systems and Operating Systems
weissel@cs.fau.de

Frank Bellosa
University of Karlsruhe
System Architecture Group
bellosa@ira.uka.de

ABSTRACT

A multitude of different hard disk power management algorithms exists—applied to real systems or proposed in the literature. Energy savings can only be achieved if the hard disk is idle for a minimum period of time. These algorithms try to predict the length of each idle interval at runtime and decide whether the disk should be switched to a low-power mode or not. In this paper, we claim that there is no general-purpose policy that maximizes energy savings for every workload and present system services that dynamically switch between different, specialized power management algorithms. The operating system automatically learns which policy performs best for a specific workload. Therefore, hard disk accesses are monitored and fed into a simulator that estimates the drive's energy consumption under different low-power algorithms. In order to recognize workloads at runtime, the system additionally monitors a set of I/O-related parameters. Using techniques from machine learning, a set of rules can be derived automatically which enable a power management daemon to identify the current workload and its optimum low-power algorithm on-line. Furthermore, the user can train the system to consider application-specific performance requirements. A prototype implementation for Linux is presented and evaluated through experiments with two different hard disks.

1. INTRODUCTION

For the area of mobile, battery-powered devices, hard disks are still indispensable to meet the ever-increasing demand for storage space. Hard disks provide higher capacities, but unfortunately consume far more power and energy than alternative storage media like flash memory [26]. Therefore, power management becomes increasingly important in order to avoid draining the batteries too fast. In the area of high-performance computing and server clusters, the ever-growing demand for storage capacity has created a different problem, as the costs (in form of electricity bills) due to energy consumption and cooling are increasing fast.

Hard disks feature several low power modes which switch off parts of the electronics or mechanical components of the drive (e. g., the spindle motor). Almost all drive models support the *standby* mode, which stops the spindle motor. Entering a low-power mode and resuming to the activate state results in an overhead in time and energy which has to be accounted for by power management algorithms. The time spent in, e. g., standby mode has to exceed the *break-even time* in order for the amount of energy saved to be higher than the energy needed to perform the mode transitions. This threshold is typically between 2 to 20 seconds for most drives. Power management is usually focused on the standby mode as it provides the highest energy savings. Therefore, low-power algorithms are often referred to as "spin-down policies".

A multitude of spin-down policies has been proposed in the literature. We argue that power management has to be task-specific; their exists no algorithm that is optimal for every workload. As an example, consider a multimedia player that reads data from hard disk at constant intervals. If the period exists the break-even time, the drive can be spun down immediately after a disk access to maximize energy savings. However, for irregular access patterns, a spin-down timeout would be beneficial in order to avoid unnecessary mode transitions.

In this paper, we present an automated approach to identify task-specific power management policies that achieve maximum energy savings at runtime. The operating system continuously records hard disk accesses and monitors I/O related system parameters. Using the simulator environment *Dempsey*, the system can autonomously test different spin-down policies on the recorded trace files and derive estimations for the hard disk's energy consumption. This way, the optimal policy for a specific access pattern is automatically learned. In order to recognize access patterns, we apply techniques from machine learning in order to derive a classification algorithm that dynamically selects an appropriate spin-down algorithm at runtime. Spin-down policies can degrade application performance as mode transitions cause additional delays. If or to what degree these delays have an influence on user experience depends on the specific application and the expectations of the individual user. As a consequence, we argue that the system cannot make optimal trade-offs between energy savings and performance without additional information from the user. The approach presented in this paper allows the user to specify performance requirements of certain applications and train the system to choose appropriate spin-down policies at runtime.

With the proposed infrastructure, the tedious job of developing "general purpose" power management algorithms that behave correctly in every situation is made easier: depending on the current task, the system automatically chooses one of a set of policies that are optimized for a specific workload, application scenario or computing platform.

Many power management algorithms found in today's soft- and hardware are based on heuristics and implicit assumptions. By observing the use of the device, these policies dynamically decide when to switch between idle and low-power modes. An example is Hitachi's *Adaptive Battery Life Extender* (ABLE) technology, which was introduced by IBM in 1995 [10, 9]. ABLE estimates the time of the next hard disk command based on the frequency and the interval between I/O requests. This algorithm chooses the most efficient low-power mode based on the expected energy savings and response delays. The user can configure a limit on the response delay by specifying the deepest low-power mode. However, application scenarios can exist for which the built-in heuristics will reach wrong decisions or the implicit assumptions may not apply. As a consequence, energy can be wasted. In these cases, an adaptation, i. e., replacement or modification of the heuristics is often not feasible.

A prototype implementation for Linux is presented and evaluated with trace files and energy measurements of two hard disks—an IBM/Hitachi Microdrive (1 GB) and a 2.5-inch Travelstar 40 GN hard disk (20 GB). The proposed approach to adaptive power management is compared with the disks' internal algorithm ABLE.

In the next section, we will discuss related work. Our approach will be presented in detail in section 3, followed by an overview of the Linux implementation. In section 5, we will discuss preliminary results.

2. RELATED WORK

2.1 Hard Disk Power Management

Spin-down policies can be grouped into on-line and off-line policies. Off-line policies are assumed to be omniscient and optimal, having access to complete information on past and future hard disk accesses.

The non-adaptive *device dependent time-out* policy (DDT), which uses the break-even time of the drive as the spin-down time-out, is proven to achieve comparably high energy savings (see [18]), and its algorithm is fast, simple and storage-efficient. DDT records the time of the last hard disk access and periodically checks if the difference between the access time and the current time exceeds the break-even time. If this is the case, the hard disk is set to standby mode. It can be proven that DDT will consume at most twice as much energy as the omniscient oracle policy. If the length of an idle period as less than the break-even time, the hard disk will be kept in idle mode. As a consequence, the same amount of energy is consumed as under the oracle policy. If an idle period exceeds the break-even time, the energy consumption is at most twice as high as under oracle.

A multitude of spin down policies has been proposed in the literature [4, 7, 13, 15, 17]. They all differ in their decisions *when* to perform mode transitions. More sophisticated algorithms try to predict the timing of future requests by observing the use of the device, dynamically adapt their decision rules, involve techniques from machine learning or rely on statistical models. Lu et al. analyze and compare several hard disk power management policies with respect to the number of spin-downs, the accuracy of the prediction (i. e., the number of incorrect shutdowns), interactive performance and memory and computational requirements [18]. Policies based on time-index semi-Markov models, together with DDT, achieve the best results over all categories.

Helmbold et al. present an approach to adaptive hard disk power management based on a machine learning technique [8]. Several experts representing different spin-down policies periodically estimate the length of the next idle period. For each expert, a weight is maintained which is increased if its prediction matches the observed idle phase length. A spin-down time-out is computed as a weighted average of all experts.

While traditional power management schemes in operating systems do not distinguish different sources of requests, Lu et al. introduce an approach that uses information on concurrently running tasks as an accurate system-level model of requesters [16]. The utilization of the device and the processor are monitored for each process. A device is shut down if the overall utilization is low.

In this paper, approaches to modify the timing of disk accesses as proposed in [25] or by Papathanasiou and Scott [20] are not addressed. Additional energy savings can be achieved by grouping devices accesses, which results in increased idle times and a reduced number of mode transitions.

2.2 Workload Characterization

Several research projects investigate methods to workload classification.

Isci and Martonosi [12] present an approach to identify characteristic program phases at runtime and derive predictions on program behavior. Two key aspects of the presented phase analysis are identified: the prediction of a single value, e. g. the instructions per cycle or a compound value, and the estimation of the duration of program phases (i. e., for how long will the value prediction be valid). Short- and long-term predictions and their applications are discussed. Methods are introduced to apply duration predictions to dynamic power management in order to account for the extra costs of transitions between operating modes or processor frequency/voltage settings.

Dynamic, *phase-based* power management distinguishes different program phases at runtime [11]. Representative execution regions can be observed and identified via different features: control flow information (program counter signatures of the executed instructions) or performance characteristics (obtained from hardware counters). With live power measurements, the energy consumption of representative program phases is determined. Phase-based approaches allow to distinguish characteristic workloads at runtime and optimize the power/performance trade-off. As the power behavior is summarized by representative execution regions, large-scale simulations can be avoided.

DFVS algorithms distinguish memory- and compute-intensive workloads (or "on-chip" and "off-chip" accesses) using information from event monitoring counters [24, 3, 22].

A lot of research has been conducted in the area of workload characterization to better understand which functions or operations are performance-critical, to optimize the performance of systems and to ease capacity planning [21, 2].

The *Program Counter Access Predictor* dynamically learns the access patterns of applications and predicts when a storage device can be switched to a low-power mode to save energy [6]. The technique to use the program counter to derive a prediction was originally applied to branch prediction for high performance processors. Here, I/O operations are correlated to particular program behavior. If a long idle period is detected the program counters following the last I/O operation are recorded to be able to identify future occurrences of this program phase before the idle interval starts.

3. TRAINING AND CLASSIFICATION

3.1 Principle of Operation

Our approach to adaptive power management is presented in figure 1.

- A set of events related to hard disk I/O is monitored by the operating system. Based on this data, features are derived by computing averages, deviations etc. A new trace file is started if the idle period exceeds a specific threshold (10 minutes).
- First, the system has to be trained. Therefore, different power management algorithms are integrated into Dempsey. For each of these algorithms, the simulator replays the disk accesses and derives the drive's energy consumption. This way, the policy which minimizes the energy consumption of a specific trace file is automatically derived. The recorded features, together with the spin-down policy, are fed into the training algorithm. As a result, a classification tree is generated.
- Second, the classification tree is integrated into a power management daemon. At runtime, this daemon monitors I/O related system parameters, traverses the classification tree and identifies the spin-down policy which was found to be optimal for the current access pattern. As a consequence, hard disk power management is dynamically adapted with respect to the workload.

For the process of *supervised learning*, application runs (training data) have to be classified by specifying the preferred power management policy. Our approach is illustrated in figure 2. This training process is automated with Dempsey: hard disk access traces are fed into the simulator. As a result, the energy consumption of the hard disk executing each trace log is estimated. This process is repeated several times with Dempsey executing different spin-down policies in order to derive the policy that maximizes energy savings for the specific workload. Alternatively, the user can specify appropriate operating modes or spin-down policies for specific applications using a configuration file. With this information, the training algorithm is invoked to compute a *Classification and Regression Tree* (see next section), representing the borders of the feature space. This tree is incorporated into an on-line classification algorithm that dynamically selects the spin-down policy that is optimal for the current workload. The whole process can be performed off-line or on-line if the system is idling or on request by the user.

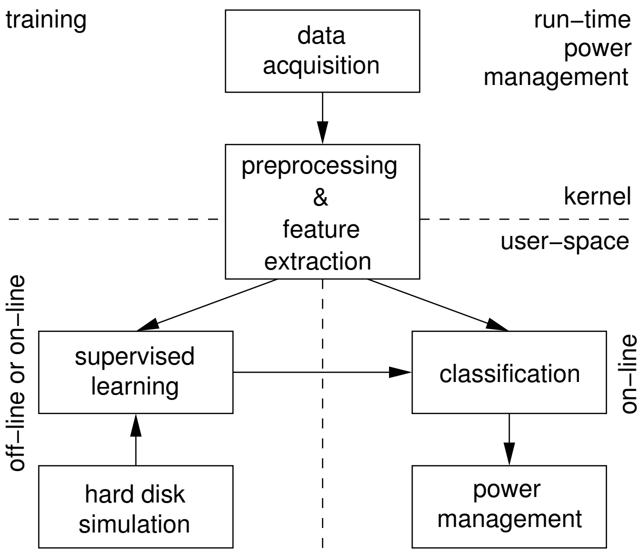

Figure 1: Training & classification: principle of operation

3.2 Hard Disk Simulation

We integrated *Dempsey* by Zedlewski et al. [26] into our power management infrastructure. Dempsey is an extension to the *DiskSim* simulator (version 2.0) [5] to estimate the energy consumed by executing a trace file of disk accesses. Therefore, in addition to performance characteristics, the power consumption of the operating modes of the specific hard disk drive have to be known.

In order to extract the power characteristics of a specific drive, Dempsey provides a bunch of C++ programs that access a multimeter via the serial port. We are currently adapting these programs to run with our own measurement hardware. For the tests in this paper, we determined

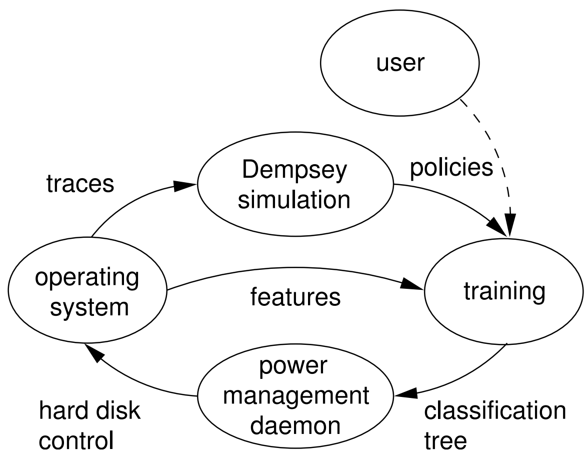

Figure 2: Supervised learning

the power characteristics using manually triggered measurements. An automated solution would definitely ease this process.

Dempsey is rather fast: on a 2 GHz machine, it takes approximately 100 ms to estimate the energy consumption of 1000 s of disk accesses.

3.3 Classification and Regression Trees

Classification algorithms have to assign observed patterns or features to classes. Classification and Regression Trees (CART) introduced by Breiman et al. [1] base these decisions on answers to binary questions. Questions are asked to arbitrary elements of the feature vector, e. g.:

```
if (average number of disk reads per time window) < 5
```

The questions are ordered in a tree structure. The first question forms the root node. Each answer to this question represents an edge to the next level of nodes and questions. The leafs of the tree represent the classes.

The tree is traversed from the root in order to classify a feature vector. The answer to a question directs the classification algorithm to the next sub tree. Questions are processed until a leaf, representing a class, is reached.

A quality factor is needed to define the order of questions. We chose the impurity of a set, defined by [14]: a set is pure if all elements belong to the same class. Impurity is maximal for uniformly distributed classes. A measure for purity is the entropy of sets according to [19]:

$$H(S) = -\sum_i P(i|S) \log_2 P(i|S)$$

This equation is only valid for uniform costs of classification errors. $P(i|S)$ is the percentage of class i in set S.

The tree is built as follows. All feature vectors are assigned to the root of the tree. Then the best question according to the quality factor is chosen. This question is used for splitting the set into two parts of maximal purity. Recursively, for each of the resulting new nodes, the best question is identified and the subset, again, is split into two parts. This process continues until all elements of each node belong to the same class or until the improvement of the error rate or the number of elements per node is below some threshold. A positive side-effect of taking the best question first and then successively the best questions for each subset is that features are already ordered by their significance: features used near the root are superior to features used at deeper levels.

3.4 On-line Classification

Based on the events monitored by the kernel, several different features can be derived, using averages, standard deviations and differences over a sliding time window of 10 seconds. In our implementation, 12 different events are captured by the operating system, resulting in a large number of features. Only a subset of all possible features is used for classification in order to avoid the effect of over-training and to keep the overhead of a runtime classification to a reasonable level. Using the training algorithm, the most significant features—the features that lead to the highest purity of each subset—are automatically identified.

Number of disk accesses
Number of disk reads
Number of disk writes
Amount of data read or written
Amount of data read (bytes)
Amount of data written (bytes)
Number of syscall invocations to read or write data
Number of syscall invocations to read data
Number of syscall invocations to write data
Average time between two hard disk accesses
Average time between two read operations
Average time between two write operations

Table 1: Features used for classification

Table 1 shows the subset of features used for classification of the hard disk spin-down policy. The numbers of disk accesses and the amount of data read or written are per time window, i. e., the differences between the first and the last value in the sliding time window are computed.

The time to react to changed resource usage, that is the time the system needs to recognize the start, end or switch to another workload, is influenced by the length of the time window over which the features are computed. A short window of only a few seconds results in a fast speed of adaptation of the power management algorithm. In contrast to that, short variations in the hard disk access pattern are smoothed out over larger time windows and the low-power policy gains more stability. For our tests, we chose a value of 10 s which turned out to be a good compromise between these two diametrical effects.

Classification and Regression Trees are implemented as a sequence of if-statements, comparing the processed features with thresholds representing class borders. The if-cascade maps the features to classes.

4. IMPLEMENTATION

Altogether, 12 events from different levels in the operating system are distinguished. We added hooks to the system calls that read data from or write data to the hard disk (`read()` and `write()` with the variants `readv()` and `writev()`). In addition to that, the time between I/O requests is recorded.

The amount of data read and written and the number of disk accesses is captured in the block device driver switch (`generic_make_request()`). This information is also used to create disk access traces.

The kernel captures I/O-related information in ring buffers. A system call is provided to retrieve this data from the kernel, flushing the ring buffer. In section 5.4, we discuss an extended kernel service and interface that allows to distinguish hard disk access patterns of different tasks running concurrently. A user land daemon is responsible for further processing of the collected events, replaying hard disk accesses in the simulator, deriving features and classifying workloads. Data is retrieved from the kernel every 100 ms. Access logs are maintained in files on a ramdisk. These logs are stored in the format "time device sector size flags", which is also used by DiskSim.

Dempsey computes the energy consumption of a hard disk through replaying a disk access log in the simulator DiskSim. Spin-down policies can be implemented in the module `disksim_power`. Policies with fixed spin-down time-outs are already supported. As on-going work, we are currently implementing more sophisticated policies, e. g., the adaptive learning tree algorithm proposed by Lu and De Micheli [18]. The actual spin-down policy to be used by Dempsey can be specified through a command line parameter. A Perl script was written that invokes Dempsey to simulate a set of trace files under different power management policies. This program records the output of the simulator (the total energy consumption), identifies the policy which maximizes energy savings for a given disk trace and creates configuration files for the training algorithm.

Next, the training algorithm is invoked. These routines are based on the *Edinburgh Speech Tools Library*, a library of C++ classes and utility programs frequently used in speech recognition software[1]. Classification and Regression Trees are implemented as a sequence of if-statements, comparing the processed features with thresholds representing class borders. The if-cascade maps the features to classes. The resulting classification and regression tree is converted into Perl code and integrated into another Perl script (`classify.pl`). This program is used for runtime classification and power management: it periodically queries the kernel to retrieve I/O-related parameters, computes the features used for classification, invokes the classification tree and activates the identified spin-down policy. The policies used in our experiments are implemented in the IDE device driver and can be selected through the `/proc` filesystem.

5. EVALUATION

5.1 Spin-Down Policies

As a first approach, we implemented a group of simple power management policies with fixed time-outs ranging from 0 to 2 seconds. Dempsey already supports this type of spin-down policies.

The Microdrive is connected to a PC via an extender card to measure the power consumption. The extender card allows the isolation of the power buses, so we attached a 4-terminal precision resistor of 100 mOhm to the 5 V supply line. Analogously, a 50 mOhm resistor is put in the power lines to the Travelstar hard disk. The voltage drop at the sense resistor was measured with an A/D-converter at 5000 samples per second and a resolution of 256 steps. Tables 2 and 3 list the energy characteristics of the hard disks used in our tests.

Figures 3 and 4 show the energy consumption of different tasks under a variety of spin-down policies. Besides tests with disabled power management ("always-idle") and the

[1] see `http://www.cstr.ed.ac.uk/projects/speech_tools`

mode	power
performance idle	839 mW
low-power idle	333 mW
standby	91 mW

transition	energy	time
standby → performance idle	721 mJ	792 ms
performance idle → standby	360 mJ	330 ms

break-even time = 1.94 s

Table 2: Energy characteristics of the IBM/Hitachi Microdrive (1 GB); 5 V power supply.

mode	power
performance idle	1.59 W
low-power idle	730 mW
standby	220 mW

transition	energy	time
standby → performance idle	3336 mJ	1305 ms
performance idle → standby	792 mJ	338 ms

break-even time = 2.75 s

Table 3: Energy characteristics of the IBM/Hitachi Travelstar 40 GN (20 GB)

internal, adaptive algorithm, we evaluated fixed spin-down policies with different time-outs: 0 s, i. e., switch to standby mode immediately after a disk access, 1 s and 2 s. For all tests with the Microdrive, the energy consumption increases with time-outs of 3 s or longer (i. e., time-outs that exceed the break-even time). Analogously, energy savings cannot be achieved for the Travelstar drive with time-outs longer than 3 s. The drive's built-in, adaptive power management policy ("ABLE") was configured to minimize power consumption by setting it to the most aggressive level. The results demonstrate that even in this mode, the policy is less efficient (regarding energy savings) than the fixed time-out rules for almost all workloads. In the following, we report on results of test runs on the Microdrive (figure 3).

First, we tested a Linux kernel compile job. We executed `gcc` 3.4 on the modified kernel of our prototype implementation (version 2.6.4). It can be seen that the always-idle policy achieves the highest energy savings. Power management has an extreme effect when running `gcc` with the immediate spin-down policy. In this case, the runtime is increased from 452 s to over 1060 s. The reason for this dramatic slowdown is that hard disk accesses arrive at intervals of less than 1 s, while the disk needs more time to spin down and up again. A lot of time is spent waiting for the hard disk to become active. As a consequence, the execution time of the compilation process is increased. The power consumption during the first 25 seconds is shown in figure 5: after the startup sequence, the disk switches frequently between idle and standby mode. If the internal, adaptive power management algorithm is active, the disk spins down only once during the whole kernel compile run. With a time-out of 2 s, the number of mode transitions is increased to eight and up to 14 if the time-out is set to 1 second.

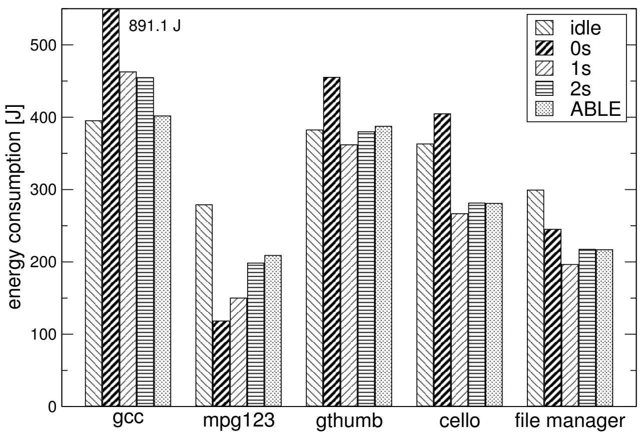

Figure 3: Energy consumption of different tests on a 1 GB Microdrive hard disk.

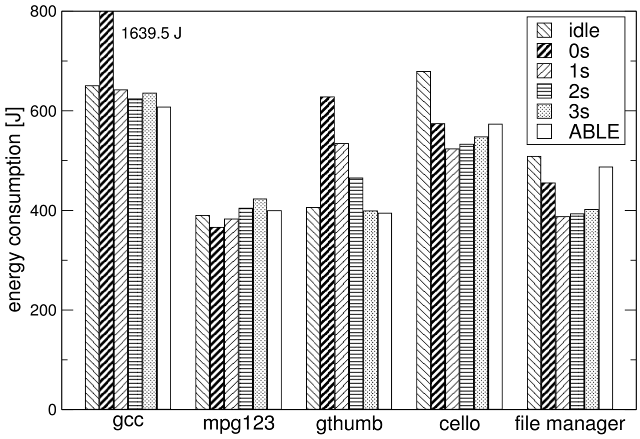

Figure 4: Energy consumption of different tests on a 20 GB Travelstar hard disk.

Next, we recorded the energy consumption of `mpg123` playing a MP3 file (128 kbit/s) from hard disk. For this workload, the immediate spin-down policy achieves maximum energy savings. The execution time of this test is 536 s independent of the spin-down policy. There is no impact of hard disk power management on the application quality, i. e., the audio playback is not delayed.

Furthermore, we ran the image viewer `gthumb` on a directory with 140 pictures from a digital camera. These pictures were viewed in slide show mode with a period of 3 seconds. In addition to that, we recorded the hard disk accesses from a user session of 10 minutes. The user worked on different directories using the file manager `nautilus`. In particular, PDF files were viewed, text documents edited and file access rights changed. For these tests, the total energy consumption was minimized when running a spin-down policy with a time-out of 1 s.

Finally, the first 10 minutes of one of the `cello` trace files from HP Labs (April 18$^{\text{th}}$, 1992) were replayed [23]. These traces were also used in the evaluation of Dempsey. Again,

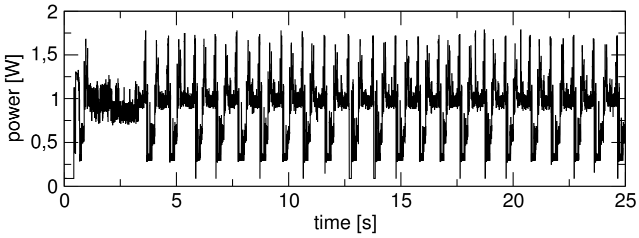

Figure 5: Power consumption of the Microdrive during a kernel compile run. The first 25 seconds are shown.

a fixed time-out of 1 s outperforms other spin-down policies.

For some tests on the Travelstar hard disk, other optimal spin-down policies were identified than for the Microdrive (see figure 4). For instance, the adaptive policy ABLE minimizes total energy consumption when running `gthumb` in slideshow mode.

5.2 Runtime Classification

In addition to the trace files of the five application scenarios discussed above, a dummy workload with no disk accesses at all and the disk's own adaptive algorithm (ABLE) as the preferred spin-down policy was used for training. The resulting classification tree that was automatically generated for the Microdrive is shown in figure 6. These rules are exported as a Perl module which can easily be incorporated into the power management daemon in user space.

We repeated the tests and recorded the classification results of the power management daemon. The classification was also evaluated with variations of the tests: `gcc` was run on the `Dempsey` source code instead of the Linux kernel, different MP3 files were played with `mpg123` and `gthumb` was tested with different slide show periods. If a new application is started, the start-up activity in the first few seconds differs from the typical runtime "behavior" of this application. In addition to that, it takes some time until the sliding time window of the classification algorithm is filled with characteristic values. As a consequence, the first 10 s of most tests were classified wrongly. In 93 % of the time, the workloads were identified correctly. The best results were obtained for the kernel compile run and the audio playback with less than 3 % wrong classifications.

5.3 User-Guided Power Management

Hard disk power management can cause additional delays due to the overhead of accelerating the spindle motor and reactivating the drive. Depending on the application, there can be an effect on the execution time or other quality-of-service aspects. For instance, we did not experience any influence of spin-down policies on the playback of MP3 files. In contrast to that, considerable delays were observed for some interactive tasks. For instance, the overhead of spin-up operations when working with the file manager `nautilus` can irritate the user. It is obvious that performance requirements depend on the specific application and the user and cannot be derived by the operating system automatically.

This issue is addressed by the proposed solution: The classi-

```
if (time between read accesses < 0.96s)
    if (time between I/O accesses < 0.66s)
        if (number of I/O syscalls < 1329)
            if (number of I/O acesses < 87)
                classify("ABLE")
            else
                classify("always-idle")
            endif
        else
            if (kbytes read < 5188)
                classify("always-idle")
            else
                classify("time-out=1s")
            endif
        endif
    else
        if (number of read accesses < 981)
            classify("time-out=0s")
        else
            classify("always-idle")
        endif
    endif
else
    if (number of read accesses < 484)
        classify("time-out=1s")
    else
        if (time between write accesses
           < 1.41s)
            classify("always-idle")
        else
            classify("time-out=1s")
        endif
    endif
endif
```

Figure 6: Classification and Regression Tree for the Microdrive hard disk. All parameters are per time window (10 s).

fication of the recorded training data can also be performed by the user. Therefore, appropriate spin-down policies or, alternatively, a limit on the performance degradation can be specified in a configuration file which is read by the training algorithm. This way, user- and application-specific power/performance trade-offs can be made at runtime.

To test this approach, "always-idle" was specified as the preferred spin-down policy for the file manager test and the kernel compile run, while the immediate spin-down was configured for `mpg123` and a fixed time-out of 1 s for the slideshow. Again, an error rate of less than 10 % resulted for the classification of the test cases.

5.4 Applications Running in Parallel

We extended the implementation to account statistics on hard disk accesses per process. Therefore, additional fields were added to the task structure. If a hard disk access is observed, the statistics of the current process are updated. The kernel interface was extended in order to allow the user land daemon to query the process ids of the tasks that issue hard disk requests and retrieve statistics on disk accesses of a specific process. This way, an appropriate spin-down policy can be identified independently for each process that operates on data on the hard disk. If different, optimal spin-down policies are determined, the power management daemon has to choose one of them that is appropriate for all applications currently active. A policy should not be activated if it increases the execution time and energy consumption of one of the tasks significantly. The simple policies used in our tests can be ordered with respect to their time-outs.

In order to evaluate the workload classification of a mixture of access patterns, we repeated the `gcc` compile run of the Linux kernel in parallel to the playback of an MP3 file from hard disk using `mpg123`. Except for the first few seconds, the compile job was correctly identified throughout the whole test run of 9 minutes. For short periods of time, the audio player was classified as `gcc`, resulting in an error rate of 4.8 % for this process. The two processes probably influence the hard disk access patterns of each other. However, the classification was sufficiently stable. While a spin-down policy achieves energy savings when running `mpg123`, the hard disk should be set to always-idle when executing the compile job. Therefore, the power management daemon left the hard disk in idle mode throughout the whole test run. As soon as the Linux kernel was built, the policy that switches to standby mode immediately after a disk access was activated.

6. CONCLUSION

In this paper, an approach to adaptive, self-learning hard disk power management is presented. The operating system learns automatically which spin-down policy achieves maximum energy savings for a specific disk access pattern. Techniques from machine learning enable the system to derive a set of rules in order to identify workloads and their optimum low-power algorithm at runtime. A prototype implementation for the Linux kernel is presented. Preliminary results demonstrate that a runtime classification of the hard disk workload is feasible. We are currently applying this approach to power management of other system components like the wireless network interface and the CPU. As future work, more sophisticated, application-specific spin-down policies can be examined that reorder or group hard disk operations.

7. ACKNOWLEDGEMENTS

The anonymous reviewers helped us to improve this paper with their useful feedback.

8. REFERENCES

[1] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Monterey, 1984.

[2] M. Calzarossa, L. Massari, and D. Tessera. Workload characterization issues and methodologies. In G. Haring, C. Lindemann, and M. Reiser, editors, *Performance Evaluation: Origins and Directions*, pages 459–482. Springer-Verlag, 2000.

[3] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'04)*, August 2004.

[4] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spindown policies for mobile computers. In *Proceedings of the Second USENIX Symposium on Mobile and Location Independent Computing*, Apr 1995.

[5] G. Ganger, B. Worthington, and Y. Patt. The DiskSim simulation environment version 2.0 reference manual, December 1999.

[6] C. Gniady, A. R. Butt, Y. C. Hu, and Y.-H. Lu. Program counter-based prediction techniques for dynamic power management. *IEEE Transactions on Computers*, 55(6):641–658, June 2006.

[7] P. Greenawalt. Modeling power management for hard disks. In *Proceedings of the Symposium on Modeling and Simulation of Computer and Telecommunication Systems*, January 1994.

[8] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the Second Annual International Conference on Mobile Computing and Networking (MOBICOM'96)*, pages 130–142, 1996.

[9] W. F. Heybruck. Enhanced adaptive battery life extender (ABLE). White Paper. Hitachi Global Storage Technologies, November 2005.

[10] IBM. Adaptive power management for mobile hard drives. White Paper, January 99.

[11] C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture (HPCA'06)*, February 2006.

[12] C. Isci, M. Martonosi, and A. Buyuktosunoglu. Long-term workload phases: Duration predictions and applications applications to dvfs. *IEEE Micro*, 25(5):39–51, September 2005.

[13] P. Krishnan, P. Lon, and J. S. Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. *Algorithmica*, 23(1):31–56, 1999.

[14] R. Kuhn. *Keyword Classification Trees for Speech Understanding Systems*. PhD thesis, School of Computer Science, McGill University, Montreal, 1993.

[15] K. Li, R. Kumpf, P. Horton, and T. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the USENIX Winter 1994 Technical Conference*, January 1994.

[16] Y.-H. Lu, L. Benini, and G. D. Micheli. Operating-system directed power reduction. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'00)*, pages 37–42, July 2000.

[17] Y.-H. Lu and G. D. Micheli. Adaptive hard disk power management on personal computers. In *Proceedings of the IEEE Great Lakes Symposium*, pages 50–53, March 1999.

[18] Y.-H. Lu and G. D. Micheli. Comparing system-level power management policies. *IEEE Design & Test of Computers special issue on Dynamic Power Management of Electronic Systems*, pages 10–19, March/April 2001.

[19] D. M. Magerman. *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University, February 1994.

[20] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *Proceedings of the 2004 USENIX Annual Technical Conference*, pages 255–268, June 2004.

[21] O. I. Pentakalos, D. A. Menascé, and Y. Yesha. Automated clustering-based workload characterization. In *Proceedings of the 5th NASA Goddard Mass Storage Systems and Technologies Conference*, September 1996.

[22] C. Poellabauer, L. Singleton, and K. Schwan. Feedback-based dynamic frequency scaling for memory-bound real-time applications. In *Proceedings of the Eleventh Real-Time and Embedded Technology and Applications Symposium (RTAS'05)*, March 2005.

[23] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Proceedings of the Winter USENIX Conference*, pages 405–420, January 1993.

[24] A. Weissel and F. Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02)*, October 2002.

[25] 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 OSDI'2002*, December 2002.

[26] F. Zheng, N. Garg, S. Sobti, C. Zhang, R. Joseph, A. Krishnamurthy, and R. Wang. Considering the energy consumption of mobile storage alternatives. In *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'03)*, October 2003.