Anwendungsspezifische Energieverwaltung in Betriebssystemen

Diplomarbeit im Fach Informatik

vorgelegt von
Thomas Weinlein
geboren am 1. Februar 1979 in Lichtenfels

Institut für Informatik,
Lehrstuhl für Verteilte Systeme und Betriebssysteme,
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer:	Dipl.-Inf. Andreas Weißel
	Prof. Dr.-Ing. Frank Bellosa
	Prof. Dr.-Ing. Wolfgang Schröder-Preikschat

Beginn der Arbeit:	15. Juli 2004
Abgabedatum:	17. Januar 2004

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 13. Januar 2005

Application-Specific Energy Management in Operating Systems

Diploma Thesis

by

Thomas Weinlein

born February 1st 1979 in Lichtenfels

Department of Computer Science,
Distributed Systems and Operating Systems,
University of Erlangen-Nürnberg

Advisors: Dipl.-Inf. Andreas Weißel
Prof. Dr.-Ing. Frank Bellosa
Prof. Dr.-Ing. Wolfgang Schröder-Preikschat

Begin: July 15th, 2004
Submission: January 17th, 2004

Abstract

Power management is recognized as an important research area for mobile devices, embedded systems and general purpose systems. There are several methods for reducing the energy consumption of individual components and of the whole system. But known methods often have the shortcoming that energy savings cause performance reduction. Therefore power management has to adapt to the applications' and users' performance demands. However there is a trade-off between the users' performance demands and energy savings. This work presents an approach to automatically identify the performance demands of applications at runtime and this way guiding the power management policies.

This approach extends the student thesis of Matthias Faerber to system wide power management and several shortcomings are addressed. In the previous work the currently running application is identified; Here, application usage profiles are classified because one application can have different usage characteristics depending on its current job. Furthermore the heuristic classification done by Faerber is replaced by a theoretically sound classification and training algorithm based on Classification And Regression Trees.

The presented approach gives the user the possibility to specify his personal minimal performance demands for different application usage profiles. Then multiple resource usage characteristics are retrieved from the CPU, the wireless network interface card and the hard disk for each application. Upon those usage characteristics it is possible to identify the usage profile of the currently running application and to dynamically apply the adequate user-defined power management setting. To account usage statistics for each application, the abstraction of resource containers is used. The mapping of resource usage characteristics to the appropriate power management setting is done by a classification demon. This

classifier is trained by supervised learning with the Classification And Regression Tree algorithm.

The proposed approach for adaptive power management was evaluated on an iPAQ running a modified Linux kernel and the classification demon. Several applications that are typical for such a mobile platform were tested. A rate of correct classifications of the user-specific performance demands and the corresponding power management settings of approximately 98% was achieved.

Contents

Chapter 1

Introduction

The demand for more powerful mobile devices, such as PDAs, cell phones or laptops, has strongly increased over the last years. The additional features and power of these devices come with high energy consumption. However the improvement of battery capacity for such devices could not keep up with that development as shown in Figure 1.1. Therefore more and more aggressive power management policies have to be introduced to keep the battery runtime on a constant level. Modern components for mobile devices bring along several power saving modes to relieve those efforts. For example Intel released the XScale processor series, which supports CPU frequency and voltage scaling. The standard for wireless network interface cards, IEEE 802.11, defines power saving modes and current hard disk drives also provide mechanisms to save energy.

1.1 Motivation

Power management policies for mobile devices result in a trade-off between battery runtime or energy savings and performance. In the majority of cases, this trade-off can be visualized as a reciprocal curve as shown in Figure 1.2.

It can easily be seen that high energy savings can result in poor performance and high performance results in little energy savings. While it is impossible to get below that curve, a power management policy has to find the optimal position on that curve. This position is individual for each application and possibly each user

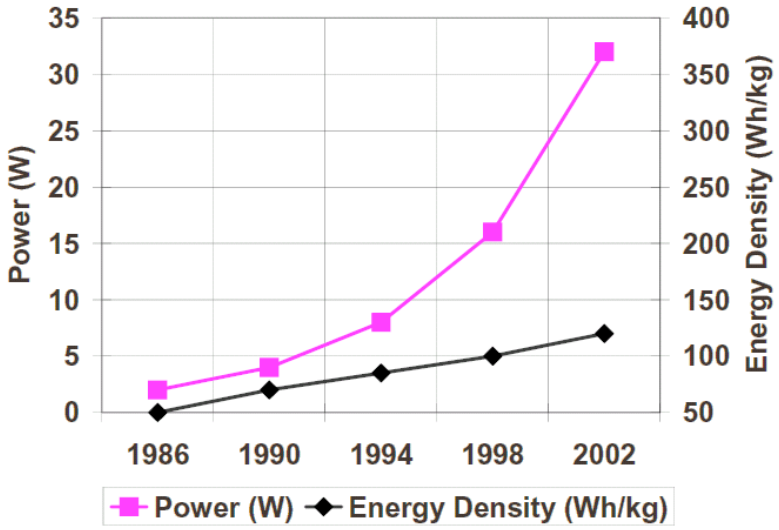

Figure 1.1: A widening gap between power requirements and the energy density of batteries [17]

because the user-experienced performance and/or the applications' performance demands often differ from the available absolute system performance.

However, most existing power saving techniques do not consider the performance demands of currently running applications but introduce a static, system wide energy and performance limit. A common example is the access delay of hard disks, which are shut down to save energy. For limiting such effects, most methods introduce a global performance loss boundary. This can be seen as a shortcoming as various applications and/or their users have individual performance demands which can be both higher and lower as the global boundary. For example a media-player playing an audio file will periodically read data from the hard disk, so that frequent switching between power states would waste more energy than keeping the hard disk in active mode. Other applications like a PDF viewer may not access the disk for minutes while the user is reading, which makes switching to standby mode profitable.

To address those shortcomings, a power management algorithm has to respect the users' performance preferences for different applications. This can be achieved by identifying those performance demands during runtime and adapting the power management settings.

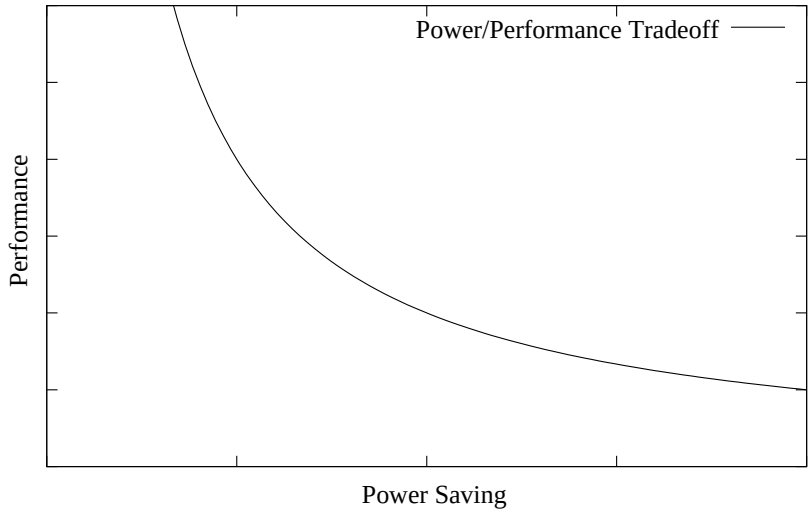

Figure 1.2: Tradeoff between power saving and performance

This work presents an approach to identify the performance demands that are associated with the currently running application or application usage profile. As a consequence it is possible to achieve the user-defined power/performance tradeoff of each application.

The approach is implemented on an iPAQ with a PXA250 CPU featuring frequency scaling, a wireless network interface and an IBM microdrive. A modified version of the Familiar Linux distribution from [13] was used and typical applications for mobile devices were tested. Such a system can be seen in Figure 1.3.

This work is based on a student thesis about application-specific power management for wireless networks presented by Matthias Faerber [9].

In his work, Faerber argues that power management has to adapt to applications' and users' demands. Thus, a system has been implemented that maps statistical usage information of the network card to a usage profile appropriate to the currently running application.

A collector module was added to the Linux kernel. It is responsible for periodically retrieving data from the network interface. Such data is e. g. the number of received and transmitted packets and their size in bytes. The collector module passes that data from the kernel space to user space through the */proc*-filesystem.

In user space, a characterization module is run to map the characteristics retrieved from the network interface to the usage profile representing a power man-

Figure 1.3: Opie Familiar Linux running on an iPAQ

agement setting. The possible power management settings are based on the variation of the beacon period of the *Power Save* mode defined in the IEEE 802.11 standard. For more information on power management modes of the IEEE 802.11 standard, see Section 2.1.2. Upon the characteristics retrieved from the collector module several key figures, including the average and standard deviation of the size of transmitted/received packets and the ratio of inactive to active periods, are calculated. These key figures are analyzed by hand to extract a representative value of each key figure for each profile. Then the characterization module is build based on this analysis.

The classification is done as follows: First, the profile of the nearest representative value is determined for each key figure of the currently running application. If one of the profiles got a majority, a new power management decision is achieved; otherwise the current setting is retained.

In this work, this approach is extended to system wide power management and several shortcomings are addressed. One shortcoming of Faerber's approach is that it can not handle applications running in parallel because of the global data acquisition. The mixed data can not be handled accurately by the characterization demon as it would lead to varying decisions. In this work, this is addressed by the introduction of the abstraction resource container to gather data for each applica-

tion separately. As a consequence, the classifier knows about several applications and can make separate decisions, which are treated by an extra logic to reach a global decision for all applications running in parallel.

Another problem is the classifier. While Faerber analyzes the key figure data by hand, a more sophisticated and theoretically sounded training algorithm, called Classification And Regression Trees, is used in this work. Furthermore, in the earlier approach the parameters for the classification decisions are hard coded in the characterization module. The classification demon proposed here uses an extra Perl module for classification, which is generated dynamically by the training algorithm. This has the advantage that the classification demon has only to be restarted to use the new classifier. Therefore, this approach can be easily used as only the favored power management settings for the applications and/or recorded data must be specified and everything else can be done automatically.

1.2 Overview

This thesis is structured as follows:

In *Chapter 2,* related approaches to power management for CPU, wireless network, hard disk and for the whole system are presented.

Chapter 3 describes the mechanisms of resource containers. In *Chapter 4* the Classification And Regression trees (CART) are introduced. Starting from the basics of the classification problem, the principles of the CART training algorithm are explained.

The implementation of the presented approach is figured out in *Chapter 5*. After showing the procedure of acquiring characteristic data of the considered resources from the kernel, the preprocessing and classification of the data in user space is explained. Another important topic of this chapter is the generation of the classification demon.

Chapter 6 presents the evaluation results done in on-line and off-line mode. The effects of running applications in parallel on the classification results are also examined.

Chapter 7 gives some proposals for future investigations and *Chapter 8* completes the work with a conclusion.

Chapter 2

Related Work

There are several research projects addressing power management for mobile devices. They can be split into two groups, power management approaches on particular components or resources and approaches that deal with system wide power management. Other projects introduce an additional API so that the applications can provide information about their future resource usage.

First, the methods for particular devices such as CPU, wireless network and hard disk, which are also covered in this work, are presented. Then a discussion of system wide power management methods follows in Section 2.2. Additional APIs for power management are discussed in Section 2.3.

Although workload classification is a new topic in connection with power management, there are several works that use it for capacity planning. These are presented in Section 2.4.

2.1 Power Management for a Single Resource

2.1.1 CPU Frequency and Voltage Scaling

The power P of the CPU depends on the operation frequency f, the voltage V and the capacity C of the circuit. Additionally, there exists an architecture-dependent constant power P_{other}. The power can be computed as denoted in Equation 2.1.

$$P = V^2 * f * C + P_{other} \tag{2.1}$$

While the capacity of the circuit is a constant, some CPUs, like the Intel XScale series, provide the possibility to run at different frequency levels. For lower frequencies also the voltage can be reduced, resulting in a quadratic reduction of energy. This can be done without reduced performance as long as all deadlines can still be met. There are different measures for the calculation of deadlines, such as response time, quality-of-service or performance. For interactive applications, the response time is an important measure; it should not exceed the perception threshold of humans in the range of 50-100ms. Further on, multimedia applications should reach a certain quality-of-service, which is often measured in the number of frames per seconds. The computation of the appropriate CPU speed to reach a deadline is not difficult, but is the determination of deadlines. These problems are addressed in the following works.

Neufeld et al. [22] have evaluated several proposed policies for setting the clock speed appropriate to the current processor usage. The investigated policies are presented by Weiser [27], Pering [25] and Govil [12]. Among them is a policy called PAST and its generalization AVGn, which try to predict the future CPU utilization by calculating the average utilization of the past. Upon the predicted utilization the appropriate clock speed for the next time step can be calculated, so that the work can be finished at the next time slice. The authors show that none of the examined scheduling policies is able to set the appropriate speed for a MPEG player. The best of the tested algorithms is PAST, which is configured in a way that only the highest or the lowest CPU speed is possible. While the optimal CPU speed for MPEG playing lies in between these borders the scheduling policy results in multiple clock speed switching. Hence, there is a lot of switching overhead in time and energy, because a clock speed change needs some hundred milliseconds of time and some amount of additional energy. This scenario could benefit from application-specific power management, which determines the appropriate speed based on usage characteristics and avoids excessive switching.

Flautner et al. [10] present a multi layer system for performance estimation called Vertigo, which is comprised of several power management algorithms. On the top, a high level algorithm ensures that users of interactive applications do not experience performance loss. Therefore, the communication between the X-server and the application GUI is observed to determine the length of interactive

periods. The clock speed is set in a way that the interactive period is processed before the human perception threshold exceeds. On the bottom level, an algorithm similar to that evaluated by Neufeld et al. is used to predict the future processor utilization by referring to the past utilization.

2.1.2 Wireless Network Power Management

The standard for wireless network interfaces, IEEE 802.11, was created having power management already in mind. Therefore, a power management mechanism is anchored in this standard. There are two power states defined for the wireless network interface:

- *Awake*: Consuming full power; transmission and reception possible
- *Doze*: Very low power; neither transmission nor reception possible

Upon these power states two power management modes are defined. In the *Active Mode (AM)*, the interface is always in *Awake* state and ready for receiving and transmitting packets. If the network interface wants to switch to *Power Save (PS)* mode, it has to inform the access point first. This is because in PS mode the interface switches to *Doze* state for the time defined by the so called beacon period. During this time, the network device is not able to receive data from the access point. Hence, data packets sent from the access point during this period would get lost. After each beacon period the network interface awakes for a short time and listens for the traffic indication map (TIM) sent by the access point to inform about buffered packets intended for the network device. After analyzing the TIM, the network device can poll the access point for the buffered data. An example of two wireless network interfaces at PS mode at different beacon intervals and their access point is shown in Figure 2.1.

There are several protocols that adapt the beacon period or time-out threshold to the current usage characteristic of the wireless network interface. As those algorithms use the past utilization to predict the future they are often called *history based* algorithms.

Krashinski and Balakrishnan [15] present a power management protocol called Bounded Slowdown (BSD) protocol that minimizes the energy consump-

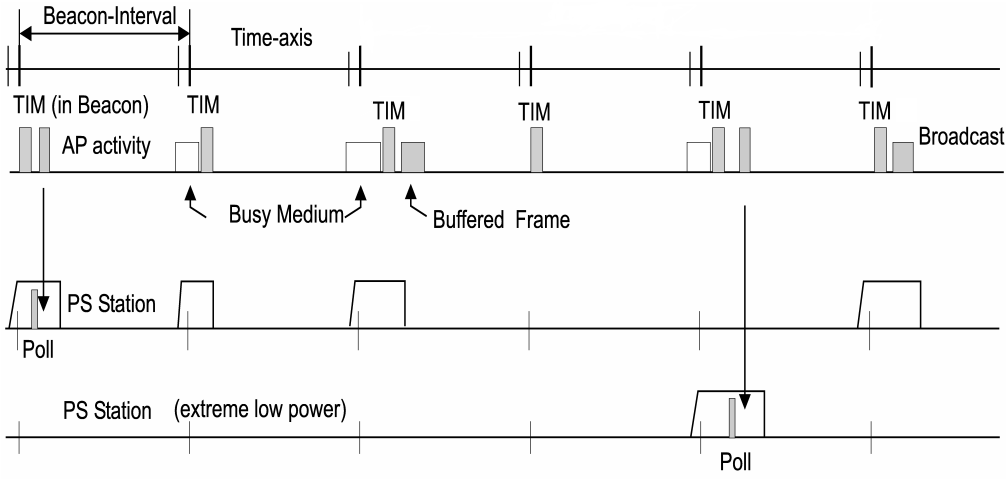

Figure 2.1: Reception of packets for wireless network interfaces working in *Power Save* mode, from [7]

tion for a guaranteed round trip time (RTT). While this method only uses information about time of transmitting and receiving packets it can be implemented in hardware.

Several proposals for optimizing the power management for multimedia streams are published by Chandra and Vahdat [5, 6]. For multimedia streams such as Windows Media, Real and Quicktime, the network packets are often sent in constant intervals. Therefore the next idle interval can be derived from the average of past idle intervals [5]. As not all multimedia streams feature a constant transmission interval, in [6] a traffic shaping method is proposed for streaming servers to ease the prediction of idle intervals on client side and thereby save additional energy. As this approach should be accurate for coordinated client-server environments, the user at the client side has no possibility to influence the power management if he has no access to the server.

Approaches for energy efficient transport layer protocols can also be found in literature. For example Bertozzi et. al [2] utilize the TCP buffering mechanism to improve energy efficiency of the transport layer with only low performance overhead.

2.1.3 Hard Disk Power Management

The ATA standard for hard disks features several power states for saving power. In *active* mode the hard disk is spinning at full speed, the read/write heads are ready and full energy is required. *Idle* mode can save some energy by moving the read/write heads to a parking position, while everything else is kept active. This results in only little energy savings, but keeps the response time low. In *standby* mode the mechanics of the disk can be powered off until the next request to the disk. Hence, more power can be saved with the drawback of a higher response time. Finally in *sleep* mode even the electronics for request handling may be turned off. Therefore a soft or hard reset is necessary to leave that mode. This has the advantage of very low power consumption, but at the cost of a reactivation time up to 30 seconds. The operating modes consuming less energy have not only the disadvantage of operation delays but the switch from one mode to the other requires additional energy. Therefore frequent switches should be avoided. The minimal time the hard disk has to stay in standby mode to save energy is called break-even time. The break-even time depends on the power consumption of the considered hard disk.

There exist three different approaches to hard disk power management. The first class includes algorithms that use a static or dynamic time-out until the hard disk is switched to a low power mode. Another class of algorithms tries to predict the workload of the future by observing the workload of the past. Finally there are policies based on statistical workload models.

Lu et al. [20] evaluated and compared several hard disk shutdown policies. In [28] the benefit of application support for hard disk power management is demonstrated. Thereby an application can flag an I/O operation as deferrable or abortable. This information can be used by the device driver to cluster accesses to the device. As a consequence the number of power state switches can be reduced as the operating system can alter and optimize the access timing of the hard disk. Furthermore, this optimization leads to longer idle periods, which result in higher energy savings.

2.2 System Wide Power Management

One approach for implementing a system wide power management is ECOSystem, which is described in [30]. The idea is to unify the power management of different devices by introducing a system wide resource and measuring unit called *currentcy* which has to be shared by all applications. It is demonstrated that ECOSystem can limit the *currentcy* accurately to reach a target battery lifetime.

ECOSystem is implemented to control the CPU, the network interface and the hard disk. Accounting and limitation of *currentcy* is done by the abstraction of resource containers that are also used in this work. The time is split up into epochs and the available energy for the current epoch, depending on the target battery lifetime, is distributed to all applications at the beginning of the epoch. A process is only scheduled if it has *currentcy* left. When a process is running, for each time slice a certain amount of *currentcy* is removed from its account, which is administrated by resource containers. For hard disk usage *currentcy* is accounted for read and write operations. Additionally, the costs for spinning the hard disk up and down are shared by all processes which accessed the disk in the period while the disk was active. Finally, transmitting and receiving bytes over the network interface is accounted proportionally to the volume and bandwidth.

The goal of this approach is the introduction of an infrastructure for system wide power management which can be adapted to several policies. While this approach is able to accurately account and limit the spent energy, the users' performance demands are not considered. In fact it is possible to prefer some processes by specifying a higher priority which permits higher *currentcy* consumption but in the end the performance is limited by the global *currentcy* limit to reach the target battery lifetime.

2.3 Application Specific Protocols

Application specific protocols for power management introduce an additional application programming interface (API). Thereby energy aware applications, i.e. applications which implement the new API, are able to specify their future re-

source demands. Based on this information the operating system can optimize the power management to the announced resource access characteristics.

There already exists a power management API for operating systems called ACPI. ACPI is a standard released by Compaq, Intel, Microsoft, Phoenix and Toshiba in 1996 [8]. It gives the operating system full control over the power management states of the devices. This is legitimate because the operating system has comprehensive knowledge about the hardware components and their usage.

Lu et al. [18, 19] argue that application programmers should not need to care about power states of the devices and instead introduce a system call named *RequireDevice(device,type,period,wait)*. Therewith, the application can announce which resource it requires and what type of access pattern it has. The access pattern type thereby can be *periodic*, *once* or *always*. Based on the requirement information, the operating system can schedule the requests such that idle periods and busy periods are clustered. The presented approach is more abstract than ACPI and programmers do not have to care about possible power states of devices from different vendors.

The shortcoming of application-specific protocols is the need to change the applications, which can be an immense overhead.

2.4 Workload Classification

Workload classification has been a research task for capacity planning and performance modeling since the early 1970s [4]. However, the thesis of Matthias Faerber presented in Section 1.1 and this work are the first approaches to use workload or resource usage classification for power management.

An approach for automated classification of workload is proposed by Pentakalos [24]. Here an unsupervised training algorithm of a Gaussian classifier is used to separate the data. Although the algorithm is unsupervised, the number of target classes have to be specified and additionally, a small user labeled starting set is needed. As the recorded data of a mass storage system is mixed up according to their class membership, an unsupervised training algorithm is mandatory to avoid excessive labeling. In contrast data for application classification is already separated for each application, which reduces the effort of labeling for the user.

Moreover user labeling is explicitly wanted so that the user can achieve his own power/performance trade-off. Another advantage of the training algorithm used in this work (see Section 3.2) is that a form of feature selection is done implicitly and is an additional result of the training algorithm while Pentakalos has to include an extra feature selection step with supplementary costs. Indeed, it has to be mentioned that the feature selection done by CART may not be optimal. Therefore, extra examinations to optimize the set of used features are recommend.

Chapter 3

Classification

The identification of different objects by investigating the characteristics of the object is an important research area in computer science called pattern recognition. Pattern recognition deals with the mathematical and technical aspects of automatic pattern processing and/or classification [23].

The basic principles of classification and an overview of available types of classifiers are presented in the next section. This is followed by the description of Classification And Regression Trees in Section 3.2. This is the type of classifier used in this work to identify the appropriate power management setting of the current application by retrieving characteristic usage data from the resources.

3.1 Basic Principles

A general classification system is shown in Figure 3.1. The digital signal $f(x)$, also called pattern, is preprocessed, e. g. filtered, to an improved signal $h(x)$. Then features are extracted. Classification can only been done if the class borders are known. These are trained in the training stage by a learning algorithm using a training sample. Based on the training results—the class borders—the classifier assigns a class to each feature vector, which represents a pattern.

Assuming that there is already a n-dimensional feature vector $\vec{c}$, the classifier associates the feature vector with one of k classes:

$$\vec{c} \rightarrow \Omega_\kappa, \kappa \in (1, 2, \ldots, k) \tag{3.1}$$

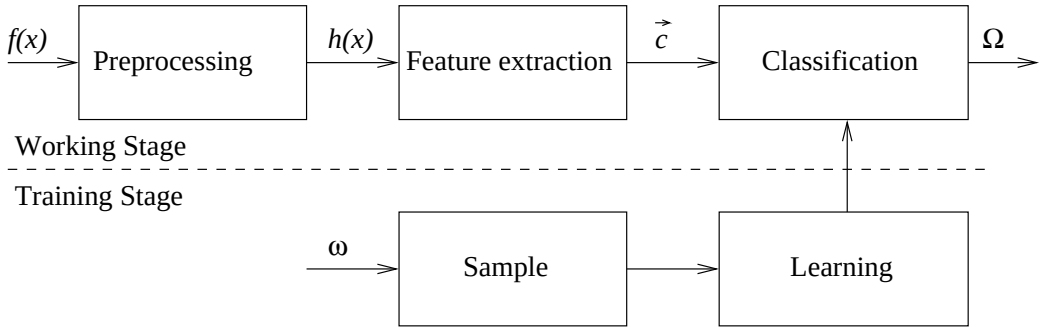

Figure 3.1: Classification system according to [23]

The optimal classifier is the Bayesian classifier. It minimizes the error rate by deciding in favor of the class that maximizes the a posteriori probability:

$$p(\Omega_\kappa \mid \vec{c}) = \frac{p_\kappa p(\vec{c} \mid \Omega_\kappa)}{p_i(\vec{c})} \tag{3.2}$$

Unfortunately, this classifier needs full statistical information. In practice, this knowledge will never be available. But there are several other classifiers which approximate the Bayesian classifier.

- *Statistical Classifier:*
  Assuming that the conditional probabilities of all feature vectors exactly belong to one n-dimensional family of densities and that the a priori probabilities of all classes are known, it is possible to estimate the parameters of the density functions for each class, based on samples. Best known member is the Gaussian Classifier.

- *Distribution Free Classifiers:*
  Here, the parameters of k partitioning functions are estimated. The partitioning functions have to be linear in parameters. This type is more general than the Statistical Classifiers.

- *Non-parametric Classifiers:*
  For this type of classifier a non-parametric estimation of density functions is done, or the next neighbor rule is used. This has the disadvantage that the whole sample has to be saved.

- *Other Classifiers:*
  There are several other classifiers which cannot be associated with the other types presented above. These are for example Classification And Regression Trees, Support Vector Machines or artificial neural nets.

3.2 Classification And Regression Trees

For now it was shown that classification algorithms have to assign each pattern to a class, but it was not mentioned how this can be done. Classification And Regression Trees (CART) [3] make those decisions based on answers to binary questions. Questions are asked considering arbitrary elements of the feature vector.

The questions are arranged 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/or questions. The leafs of the tree are labeled with class names.

To classify a feature vector, you start at the root and answer the corresponding question in consideration of the feature data. Then you follow the edge that suits to the feature vector and answer the question of that node. This is done successively until a leaf is reached, which is labeled with some class name κ. So the feature vector is assigned to class κ.

Such a classification tree is build by the following steps:

First the set of all possible questions is build. Let the feature vector be made up of random numbers $X_i, i \in 1,\ldots,n$, then a question refers exactly to one random number X_i. The questions have the form:

$$\begin{array}{lll} \text{Is } X_i \in S? & \text{with } S \subset x_1,\ldots,x_{l(i)} & \text{for discrete random numbers} \\ \text{Is } X_i \leq \Phi? & \text{with } \Phi \in (-\infty,\infty) & \text{for continuous random numbers} \end{array} \tag{3.3}$$

So there exist as much possible questions as members of the random numbers codomain. For continuous random numbers, the magnitude of questions would be infinity. To solve this problem, the codomain is split into m equal ranges, and only the borders of those ranges are used to form questions.

Now all possible questions are available, but it is still unclear which questions should be asked in which sequence. So a quality factor is needed. Here the impu-

rity of a set is chosen, defined by [16]. A set is pure, if all elements belong to the same class κ. A set is impure, if it contains elements of different classes. Impurity is maximal for uniformly distributed classes. A measure for purity is the entropy of sets according to [21]:

$$H(S) = -\sum_i P(i \mid S) \log_2 P(i \mid S) \tag{3.4}$$

Equation 3.4 is only valid for uniform costs of classification errors. $P(i \mid S)$ is the percentage of class i in set S.

Hence, we can build the tree 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. The root has got two new children, associated with the emerged subsets. These new nodes are treated like the root node, which means that for each subset the best splitting question is picked out. That process is done, until one of the following stop criteria are reached.

- All elements of the node belong to the same class.
- Improvement of error rate is below some threshold ϕ.
- Number of elements per node is below some threshold Φ.

This is a greedy algorithm for discovering a local maximum, which may not be the global maximum. It is supposed to work also well if only a local maximum is found. A positive side-effect of taking the best question first and then successively the best questions for each subset is that you get a feature selection for free. Features used near the root are better than features used in deeper levels or even not at all.

Classification And Regression Trees are susceptible to over-training, which means that the necessary generalization is lost. For example the training data is split very subtle with high purity of each subset but testing the tree with disjoint test samples yields high error rates. This effect can be reduced by pruning the tree as shown in [11]. The training data is split at random in two parts, one for training and the second much smaller one for pruning. First of all the CART is built with

the training data as shown above. Then the pruning samples are classified by the trained tree. For each leaf the purity of the pruning sample subset assigned to that leaf is computed. If for a pair of leafs, where a pair of leafs are two leafs with the same parent node, the purity of each leaf is lower than the purity of the unity set, this pair is deleted. Thus the subsets of the leafs are merged together again and the unity set is reassigned to the parent node, which becomes a leaf now. This is done for all possible pairs of leafs. In the end the Classification And Regression Tree is more general and will probably give better results for unknown samples.

Chapter 4

Resource Container

In oder to be able to account resource usage or characteristics, an infrastructure is needed to save such information. Banga [1] introduced such an abstraction called resource containers. Resource containers are handled by the system kernel, but can be accessed by an application programming interface (API) from user space. They are also used for limiting resource consumption of activities.

There already exists a structure for accounting in the kernel, the process structure. While the process is a protection domain, there is no reason to restrict accounting to a protection domain. So to achieve more flexibility and to be able to account for different granularities, an independent structure, such as resource containers, has to be introduced. This has the advantage that on the one hand accounting for multi-threaded applications is easy and on the other hand that accounting for different clients of a single threaded server is possible.

4.1 Resource Container Hierarchy

It was already mentioned that resource containers are managed independently from processes to allow accounting for different granularities. These different levels can be easily obtained because of the resource container hierarchy. This means that all resource containers are linked in a tree like structure. There is a root container representing all resources available in that system. It can be used for system wide policies. All other resource containers are descendants of the root

container. To introduce a new level in the hierarchy, resource containers can be grouped and assigned to a new parent container.

Resource containers can be created in two ways. Either they are created by explicitly using the new system call *rc_clone()* or implicitly during the generation of a child process by the system call *fork()*. Containers generated by *fork()* are automatically attached to the new child process. Explicitly created resource containers are not bound to any process.

4.2 Limitation of Resource Usage

The detailed knowledge of resource consumption obtained by resource containers can be used to limit or control the resource usage. The limitation is suitable for applying a scheduler that enforces resource limits e. g. energy consumption. Due to the resource container hierarchy, it is possible to employ different short and long term limits for a resource. For example a long term limit for energy consumption can be specified as a resource container. Here long term can be e. g. a time interval of one second and a limit of energy consumption of nJ. A child resource container may specify a short term usage rate, which may even be higher as the long term rate, to be able to deal with usage peaks. A short term can be e. g. $\frac{1}{10}s$ and the associated energy consumption $mJ > \frac{n}{10}J$. Therefore, a peak energy consumption of mJ is possible as long as the long term energy consumption limit does not exceed nJ. So all derived resource containers and the associated processes have to hold the specified limits. The scheduler has to enforce the compliance of the limits.

4.3 Representation of Resource Containers

Because resource containers are used for limiting resource consumption of activities, the right to use a resource container has to be protected. This is done by using capabilities. A capability represents the right to access a protected object. All capabilities are managed by the operating system kernel. As most operating systems use a combination of access control lists (ACL) and capabilities to protect objects,

this is also done for resource containers. An ACL is a list of users, which have the authority to access the associated object. This mixture of ACL and capability is implemented as follows. The resource containers are represented as files in a special file system. For each file exists an ACL, which defines who is allowed to open that file. As result to the open call, the initiator receives a file descriptor representing a capability which offers access to the file. Indeed resource containers are no real files, hence read or write is not possible. The resource container files are only used for associating resource containers with names.

4.4 Modifications to Resource Containers

In this work, the resource container implementation of Martin Waitz for the Linux kernel is used but the mechanisms for resource limitation were removed from the original implementation. This is because limitation is not necessary for this work and would only cause additional overhead.

Additionally, structures for storing accounted usage characteristics are added. There is a supplementary structure for the characteristics of each resource (CPU, WLAN, HD). This is necessary because resource containers per default account only one resource usage value; here, the possibility to store an arbitrary number of parameters is needed.

Chapter 5

Implementation

The adaptive power management system described in this work was implemented for a Linux system running on an iPAQ with a PXA250 CPU, a Cabletron wireless network interface and an IBM microdrive. The kernel was modified by adding resource container support originally implemented by Martin Waitz [26] and additional code for logging of several resource parameters. For application-specific power management, a set of user space tools was added supplementary to achieve the decision which power management setting is best for the currently running applications.

The following overview of the implementation is shown in Figure 5.1. *Data acquisition* in kernel space for the resources *CPU*, *WLAN* and *HD* is described in the next section. For each application, a *User space logging* process (see Section 5.2) exists, which retrieves the data from kernel space. The logged data is stored in so called *Trace Files* to be able to train the classifier off-line with Classification And Regression Trees (*CART*) as presented in Section 5.3. In Section 5.4 the *Classification Demon* is introduced, which receives the usage characteristics from the *User space logging* processes and classifies that data using the classifier trained by *CART*. After the correct usage profile class is identified, the new power management setting is applied by using the appropriate utility for each resource, e. g. *iwconfig* for the wireless network interface.

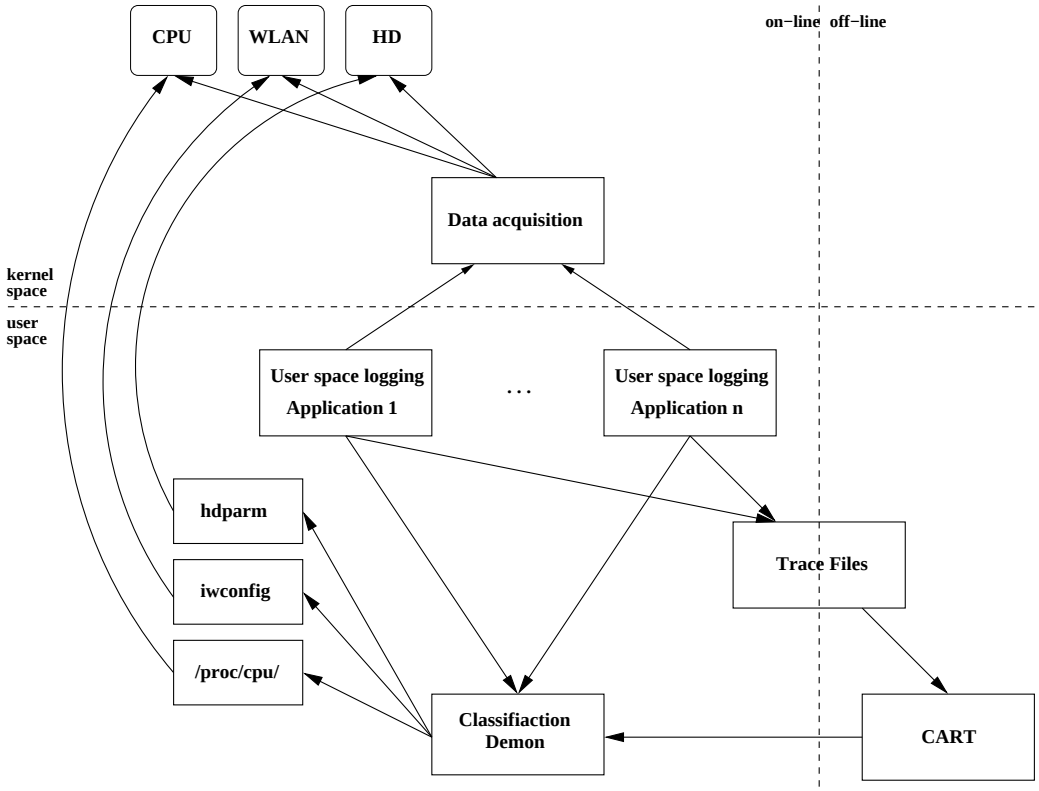

Figure 5.1: System overview

5.1 Kernel Modifications

First, resource container support was added to the Linux kernel. Resource containers provide a mechanism for accounting resource usage independent from process structures. Therefor, resource containers are arranged hierarchically to allow easy access to resource usage on arbitrary granularity. For further information on resource containers see Chapter 4.

Resource container support was added by porting the implementation of Martin Waitz for Linux kernel version 2.6 to the Linux kernel for iPAQs available at [13]. Further on, the resource container implementation was extended for accounting arbitrary information for each resource instead of only accounting the CPU usage. Therefor, the resource containers are extended by structures to log information for each resource.

The logging of resource usage characteristics is done on several levels in the

kernel. Parts of the characteristics are gathered on system call level, others are gathered on the level of the virtual file system and several on device driver level. The usage characteristics and their acquisition are presented below.

As an example, the gathering of the usage parameters of the wireless network interface is explained in detail. The number of packets transmitted or received and their size seemed to be good parameters. This means that these parameters appear to differ for applications with different performance demands and therefore should make a contribution to the identification of usage profiles. The first promising location to retrieve such information is the network device driver. The appropriate driver for the wireless network interface used in this scenario is the *wvlan_cs.o* module. The functions for sending and receiving a packet are *wvlan_tx()* and *wvlan_rx()*, respectively. So this should be the adequate locations to account the number and size of transmitted and received packets. Since this is true for transmission, the number of transmitted packets is incremented during the execution of *wlan_tx()* and at the same time the size of transmitted packets is accounted. This accounting is not done for a global variable but for the resource container associated with the currently running and sending process. The accounting to the resource container and its parents is done by the function *rc_driver_stats()* that was newly implemented for that purpose.

If you try to log the information of received packets in such a simple way you would probably account for the wrong resource container. That's because of the following reasons: The processing of received packets is triggered by an interrupt. But the principle of resource containers is to account usage for the originator of the processing, which is an interrupt in this case. The currently running process may not be the receiver of the network traffic, so the resource container of the currently running process may be the wrong one either. It should become clear that it is necessary to determine the receiver of the packet to account for the correct resource container. But to achieve this on the level of the device driver, a lot of additional processing would be necessary. To avoid such overhead, the accounting of the incoming traffic is done on the level of the transport layer because there the receiver of the packet is determined anyway.

Someone may argue that packets at the transport layer can be made up of several packets from the network layer, which leads to incommensurable values for

Logged Parameters
Status (working as client or as server or for itself)
CPU usage
CPU usage as client
CPU usage as server
CPU usage while working for itself
Active time (time of the CPU burst)
Waiting time (time of the I/O burst)
Inactive time (time in scheduling queue)
Number of packets received/transmitted
Number of bytes received/transmitted
Number of hard disk sectors read/written
Number of buffers read/written
Duration of the read or write call
Time between two read or write calls
Number of read/write calls on a file
Number of read/write calls on a socket
Number of read/write calls on a device
Number of read/written bytes of all read/write calls

Table 5.1: List of parameters logged in kernel space

the number of transmitted and received packets. The same applies to the number of bytes transmitted and received, because the packets are shortened by the length of the headers of the lower protocol layers. However, for this work it is not important to retrieve absolute correct data or information that can easily be interpreted by humans, but to retrieve data which is characteristic and distinguishable for different applications.

A list of all parameters which are logged by means of resource containers are shown in Table 5.1. The parameters of receive or transmit and of read or write operations are listed only once with the notation receive/transmit and read/write, respectively.

5.2 User Space Logging

To be considered for application-specific energy management, an application has to register at the resource container infrastructure and to communicate with the classification demon. This could be done by the application itself but would result in the need of adapting the application. As modification of applications is undesirable and often not possible another registration method was introduced. A start script was created to register an application that should be considered by the adaptive power management. The start script only needs the application name as parameter. Further on, applications are often invoked by a click on an icon of the graphical desktop environment. As the 'on click' behavior can be easily adapted for most desktop environments, it is possible to automatically invoke the start script for each application instead of the application itself. Therefore the user experiences no loss of comfort.

There would also be the possibility to integrate this functionality to the Linux binary loader. This would have the advantage that neither an additional start script nor application modifications are necessary. This was not done because of the following reasons: During development there occur several changes which can be applied to a script with less effort. Another reason is that the binary loader would also include all background tasks and demons to adaptive power management. This is not wanted because those tasks should deal with the available performance while the applications currently utilized by the user should enforce their performance demands.

When the user starts an application by calling the start script with the application name as parameter, a new resource container is created in the resource container file system with the application name as file name and the start script PID as resource container ID. Then the application itself is started and attached to the new resource container. Afterwards user space logging starts.

The resource characteristics are retrieved ten times per second through the *resource_info()* system call which was already part of the resource container implementation of Martin Waitz [26]. To be able to obtain the average and standard deviation of the characteristics such as time stamps or time intervals, these characteristics are stored as arrays in the resource containers, while characteristics

such as the number of received or transmitted packets are stored as single values. Those arrays are retrieved by the user space logging and are used to compute the averages and standard deviations since also for these characteristics only a single value is needed. On the one hand the vector of all characteristics is passed on to the classification demon and on the other hand it is written to a file for off-line processing. Logging stops when the application exits. A short cutting of an imaginary log file follows:

Counter	data[1]	data[2]	data[3]	data[4]	data[5]
4	22.1	10	32	41	32.24
5	21.2	15	23	42	23.42

Each line corresponds to one vector of all characteristics, also called data vector in the following. The characteristics are denoted as *data[1], ..., data[5]*. Additionally, there is a line counter at the beginning of each line for debugging purposes.

5.3 Building the Classifier

The classifier is built by the Classification And Regression Tree training algorithm provided by the Edinburgh Speech Tools Library [14], which is under a free license similar to the X11 license. The CART algorithm is explained in Section 3.2. It was mentioned that the CART algorithm needs a classified training set for the so called supervised learning. Therefore, the training data included in the trace files has to be labeled. Thus the implemented training script reads the name of all trace files and prompts the user for the class which it should assign to the file data. To reduce user interaction, the input is saved in a configuration file, so that the user is only asked for labeling new files.

Then the features have to be computed from the usage characteristics. To be able to calculate features which consider the usage characteristics' history, a sliding window of the k newest data vectors is used for feature calculation. Hence, all features and possible classes are defined in a global configuration file. The syntax of the configuration file has the following structure:

(
(CLASS_*resource* class0 ... classN **)**
⋮ ⋮
(*featurename* range **)** *resourceList* **;** *function* index [parameters]
⋮ ⋮ ⋮ ⋮
)

The bold face marks characters that are mandatory. Names which can be chosen by the user are written in italic type. An example of a configuration file can be found below.

First the resources (CLASS_*resource*) are specified with all possible power management class values. They are followed by the feature definitions. Therefor, a feature name and a range of values has to be specified. Possible settings are “float” and enumerations. If “float” is denoted, the CART algorithm splits the whole real number axis automatically in *m* segments; the segment borders are considered as enumeration. This is done because the CART wants to form all possible questions, which is not possible for a continuous range. For further information see Section 3.2.

The *resourceList* represents the list of resources for which that feature is taken into account by the training algorithm. After a semicolon, the method to compute the declared feature follows. Possible functions are:

- the newest value in the parameter vector
- the average of the sliding window
- the deviation of the sliding window
- the difference between the first and last value of the sliding window
- the weighted average of the whole history
- the median of the sliding window

The weighted average is computed by a weighted sum of the newest value and the weighted average of the last step. Therefore, this feature is influenced by all values and not only from those of the sliding window.

Features should have similar values if they belong to the same class and vary if they belong to different classes. Thus, averages or the median are a good idea to achieve less varying values for time dependent parameters. The deviation can be used as a measure of periodicity. To obtain local changes of a parameter the discrete derivation or difference is used. Although this are common methods for feature extraction, a rather wide range of functions can be used to compute features.

The *index* denotes which column of the data vectors is used for the feature calculation. Optional there may be further parameters for the feature calculation. For example the weighted average has an additional parameter to adjust the weight. An example of a configuration file:

```
(
( CLASS_CPU    100 200 300 400 )
( CLASS_WLAN   000 100 500   )
( foo                float   ) CPU WLAN ; average         1
( bar             10 15 20 25  ) WLAN     ; lastValue       2
( foobar             float   )           ; weightedAverage 1 0.1
)
```

For feature calculation a sliding window is used, including 40 vectors of usage characteristics. For the presented example however a window size of two is used to preserve clearness. To calculate the feature *foo* defined in the example, the average of the column data[1] is computed as the index specified for this feature is 1. The result can be seen below. For the feature *bar* the newest value of data[2] is taken, namely the value of data[2] of data vector 5 (see Section 5.2). The feature *foobar* is calculated on column data[1] again. It has to be pointed out that the feature *foobar* is calculated for all data vectors, but is never used for classification because of the empty resource list. Thus, such features should be avoided because of the overhead of feature calculation. To complete the feature vector the class labels assigned to the trace file of the considered data vectors are added. So the example results in the following feature vector, which is composed of two class labels and three calculated features:

classCPU	classWLAN	foo	bar	foobar
200	500	21.65	15	22.01

The feature data is forwarded to the Classification And Regression Tree training algorithm. The training for each resource specified in the configuration file is done separately. As the training algorithm of the Edinburgh Speech Tools Library needs separate configuration files for each resource, these are generated automatically from the global configuration file presented above. Additionally, the CART training algorithm needs parameters for the stopping criteria. As explained in Section 3.2, the CART algorithm divides the set of feature vectors into subsets of higher purity. It was also mentioned that the CART tends to over training. This effect can be reduced by specifying the minimal size of a subset, because smaller subsets imply less generality. Due to the amount of training data (about 100,000 feature vectors) a value of 500 for the minimal subset size turned out to be appropriate. A second instrument to avoid over-training is pruning. Hence, the CART also uses 10% of the training data for pruning. Referring to the current example, a possible output tree for the resource CPU and one for the network device is shown in Figure 5.2.

The output of this algorithm is printed in Lisp list syntax. As there is no Lisp but a Perl interpreter for the Familiar Linux distribution, and Perl is qualified for easily processing text or configuration files, it is used to implement the Classification Demon. Thus the tree has to be transformed to a sequence of if-statements in Perl syntax before it can be used with the classification demon written. Furthermore each if-cascade is encapsulated in a function called *classifyRESOURCE()*, where RESOURCE denotes the name of the associated resource. These functions are packed in a new Perl module which is included by the classification demon described in Section 5.4. Therefore no changes to the classification demon are necessary after new training; It only has to be restarted to automatically load the newly generated module.

5.4 Classification Demon

5.4.1 Classification

The classification demon is started on system startup, sets up a named pipe and waits for usage characteristics sent to that pipe by the users' applications. The demon can distinguish data of different applications by the process identification number which is added to each data vector sent through the pipe.

For each application, a data window is held to store the latest 40 data vectors. The recent feature vector is calculated from this data set as described in Section 5.3. Then the feature vector is classified for each considered resource by means of the generated if-cascades. The classification results are then compared to the current power management settings. But the power management setting is only changed if *n* successive classifications yield the same result. This is necessary to avoid multiple switching between different power management states in case of uncertainty of the classifier, which could result in oscillating decisions.

5.4.2 Applying the Power Management Setting

If a switch to a new power management setting is needed, the appropriate user space tools are executed in background. To apply a new setting for the CPU, the /proc-filesystem is used, which is a pseudo-filesystem that provides an interface to the kernel. While most files hold read-only information about the kernel, some files are writable to set kernel variables. Such a writable variable is the speed of the first CPU */proc/cpu/0/speed*. It can be accessed as a text file; therefore, the processor clock rate can be set e. g. as follows:

echo SPEED_IN_KHZ > /proc/cpu/0/speed

The power management of the wireless network interface card can be adjusted by the wireless tools. They provide the *iwconfig* utility to manipulate the configuration of a wireless network interface (e. g. *eth0*). The options *power* and *period* are used to specify that a new beacon period is set for power management.

iwconfig eth0 power period BEACON_PERIOD

Finally the hard disk standby time-out can be manipulated in steps of five seconds via hdparm e. g. for hard disk /dev/hda:

```
hdparm -S TIME /dev/hda
```

If several applications run in parallel, it is possible and probable that the classification demon will come to conflicting decisions for the same resource. In this implementation, the classification demon accomplishes the setting that represents higher performance. This is done due to the assumption that the setting with higher performance provides enough margin to deal with all applications running in parallel. This is surely only true for a small number of applications running in parallel but on mobile devices a maximum of two or three applications working concurrently is realistic. Hence, more sophisticated algorithms can easily be integrated.

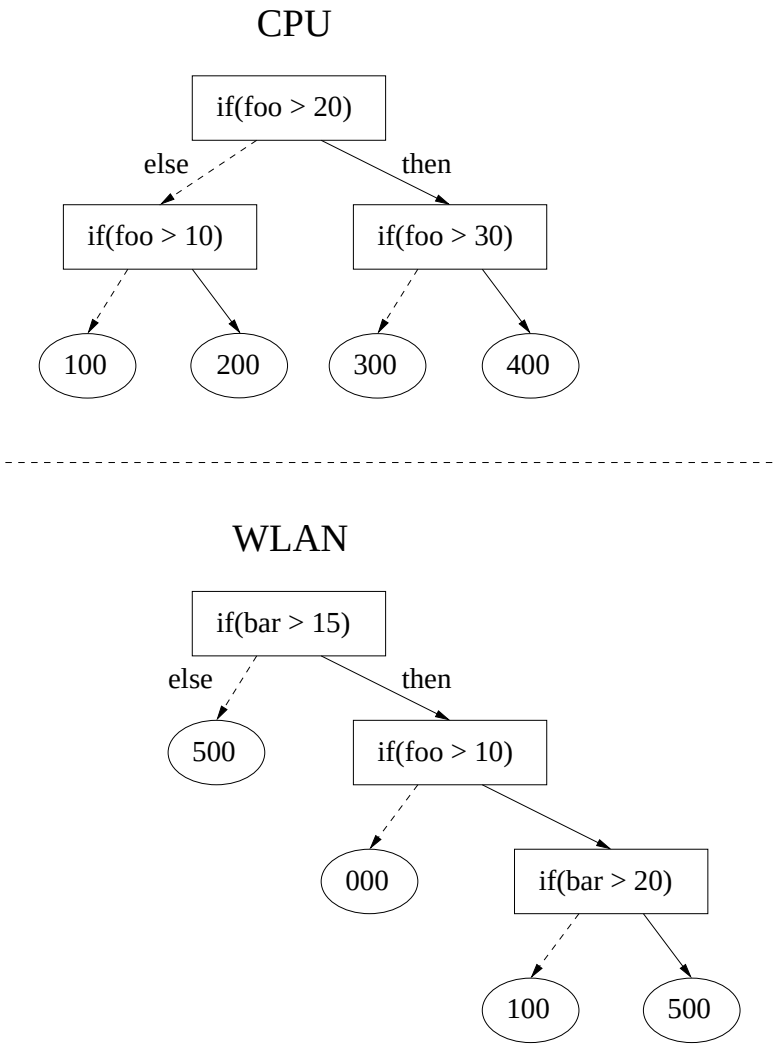

Figure 5.2: Possible CARTs for the resources CPU and WLAN, based on the given example

Chapter 6

Evaluation

The evaluation of the presented approach was done on an iPAQ with an Intel PXA250 CPU featuring frequency scaling, a Cabletron wireless network interface and an IBM microdrive. On this platform, the Familiar Linux distribution from [13] was run. The kernel was modified for additional Resource Container support and for logging of kernel characteristics. For details see Chapter 5.

6.1 Applications

Since the target platform for this work were mobile devices, the system was tested for applications which are considered as typical for this scenario. These applications can be sorted into several categories: There is software for entertainment, like games, media players and image viewers, and for use with the Internet, to browse the web, download files or access remote computers. Additionally mobile devices are often used as Personal Information Manager (PIM), for organizing appointments and for office concerns, like viewing and editing documents. The goal of application selection was to cover all presented categories to setup a representative testing environment.

The following applications were chosen for testing:

- Web browser (Konqueror)
- Media player (Opieplayer2, VLC)

  - audio and video streams
  - audio and video files
- PDF viewer (Qpdf)
- File download (scp)
- SSH session
- Game (Patience)
- Image viewer (Showimg)
- Bitmap painter (Drawpad)
- Calendar (Datebook)
- Text editor (Textedit)

These applications do not only differ in their purpose, but also in their characteristic resource usage scheme and their resource demands. Therefore, it was investigated if the resource usage characteristics, obtained as described in Section 5.2, can be used for classifying the assigned resource profile. Different CPU speeds, WLAN beacon periods and hard disk standby time-outs were distinguished.

6.2 Power Management Settings

Possible settings for CPU speed are clock rates of 200, 300 and 400MHz. For playing audio files or downloading files from remote computers a clock speed of 200MHz is sufficient, while even 400MHz is too slow to play enduring smooth video. A clock speed of 300MHz is appropriate for most interactive applications.

For the wireless network interface, the IEEE 802.11 standard defines a power saving mode. The network interface card is put to *doze* state and woken up periodically to ask the base station for new packets. The time interval for which the device is sleeping is called beacon period and can be adjusted (see Section 2.1.2).

Features
Standard deviation of I/O idle times (time between two accesses)
Standard deviation of CPU active time
Standard deviation of CPU inactive time
Weighted average of CPU active time
Standard deviation of CPU waiting time
Average of CPU active time
Total CPU time within the considered time window
Standard deviation of I/O durations (length of I/O operations)
Number of read operations from files within the considered time window
Average of CPU inactive time
Number of write operations to device within the considered time window
Number of network packets received

Table 6.1: Features used to classify for resource CPU. The features are ordered by quality.

The hard disk can be spun down to save energy. This is done after the standby time-out occurred. For this work a short time-out in the order of the break-even time and a longer one for applications accessing the hard disk more often can be chosen.

For each application, the appropriate power management setting was determined beforehand. This was done by a user who ran the applications with different settings and determined those setting that represents his personal optimal power/performance trade-off.

About three hours of training data were recorded by using the applications presented in Section 6.1. This data was labeled with the optimal power management settings and was afterwards used by the Classification And Regression Tree training algorithm to generate the in terms of purity best class borders in feature space.

Therefor, features retrieved of all resources are used as possible features for any classification tree built by CART. 36 features computed on the characteristic usage parameters which are listed in Table 5.1 were available. Hence, possible features for the resource CPU were e.g. the average active time of the CPU as well as features that are totally unrelated to the CPU such as the number of write

Features
Number of read operations on sockets within the considered time window
Standard deviation of CPU inactive time
Data volume written within the considered time window
Number of read operations from files within the considered time window
Standard deviation of CPU active time
Total number of transmitted bytes
Total number of received bytes
Average of I/O idle time (time between two I/O accesses)

Table 6.2: Features used to classify for resource WLAN. The features are ordered by quality.

operations to sockets or the standard deviation of the time between two hard disk accesses.

In Section 6.4.2 other sets of possible features are evaluated. There the set of all features is split into a subset of features related to the resource to classify and a subset of unrelated features. Then the CART does its training on each subset separately. For every other test all 36 features were used for training.

As mentioned in Section 3.2, the CART algorithm automatically selects the best features. This are the features that divide the feature space so that the highest purity for the subsets is achieved. The Tables 6.1, 6.2 and 6.3 list the best features for classifying the resource profiles of CPU, WLAN and Hard Disk. Thereby, the features are ordered such that the best is on top of the list and the worst used feature at the bottom.

6.3 On-line Evaluation

6.3.1 Resource Profiles

To evaluate this approach under realistic conditions, an on-line test was performed. The applications listed in Section 6.1 were tested on the iPAQ running the modified kernel and the classification demon. The test runs of the different applications took between 10 and 15 minutes each.

Features
Standard deviation of I/O idle times (time between two accesses) retrieved at the level of the cache subsystem
Standard deviation of I/O idle time (time between two accesses)
Average of CPU active time
Weighted average of CPU waiting time
Data volume written within the considered time window
Weighted average of CPU inactive time
Standard deviation of CPU waiting time
Average of CPU waiting time
Data volume read within the considered time window
Standard deviation of CPU active time
Total CPU time within the considered time window
Standard deviation of CPU waiting time
Total number of transmitted bytes
Number of bytes transmitted within the considered time window
Number of packets transmitted within the considered time window

Table 6.3: Features used to classify for resource hard disk. The features are ordered by quality.

The classification demon receives data from user space logging and computes features from this data. Then the calculated feature vector is classified for each resource (CPU, WLAN, HD) separately. If the classified power management settings are constant over five successive feature vectors, the classification demon initiates switching of the power management settings. To be able to evaluate the on-line test, a log file was generated which contains the timestamps and settings of each switch.

The constraint to reach five equal successive classifications before changing the power management settings was introduced because needless switching based on temporary classification errors should be avoided. An amount of five successive classifications turned out to reduce the amount of unnecessary changes of the power management settings while the response time to new performance demands is kept low.

The evaluation was done by determining the percentage of the application runtime for which the correct power management setting was chosen. This was

Application	Resource		
	CPU	**WLAN**	**HD**
konqueror	100%	99.7%	—
opieplayer			
• audio from file	96.9%	—	98.1%
• video from file	98.8%	—	99.3%
• audio stream	99.5%	93.1%	—
• video stream	71.8%	100%	—
qpdf	100%	—	100%
ssh	100%	100%	—
scp	100%	99.3%	99.0%
showimg	100%	—	100%
patience	97.7%	—	—

Table 6.4: Rates of correct power management settings

done for each resource separately.

When an application is started, the startup activity in the first few seconds differs from the typical characteristics of this application. Additionally time passes until the sliding window of the classification algorithm is filled with typical data. Therefore, the first 15 seconds were ignored for computing the rate of correct power management decisions. The results are shown in Table 6.4. Almost every application was classified correctly, which results in an average rate of correct classifications of about 98%.

The only test run that is worse is the opieplayer playing a video stream. A reason could be that the video stream for the on-line test had a lower bit rate than most of the video streams used for training. As video streams with low bit rates cause less computing, the classifier decided in favor of a lower CPU speed as specified.

The computing overhead of this approach can be measured by means of the resource containers. The overhead of the classification demon is in the order of 0.01%. Additionally, there is an overhead in the order of 0.001% for each running application resulting from user space logging. Therefore, the overhead can be neglected.

Application	Resource		
	CPU	WLAN	HD
opieplayer audio stream	96.9%	98.1%	—
qpdf	100%	—	100%

Table 6.5: Rates of correct power management settings of applications running in parallel

6.3.2 Applications Running in Parallel

Besides good classification rates for each single application, another requirement of a power management algorithm is the ability to deal with applications running in parallel. When the algorithm collects system wide characteristics without assigning them to a concrete application, the characteristic data of applications running in parallel is mixed together. This may not be a problem if the applications have similar resource demands and power management settings but otherwise this leads to multiple switches of settings [29]. In such cases the classification is done as if the data originates from a single application but in fact data segments of both applications are classified alternately. Thus the classification results alternate either.

Therefore, it has to be possible to assign usage characteristics to applications to achieve accurate results. This can be achieved by means of the resource container architecture introduced in Chapter 4. This was realized in this work as explained in Section 5.

Hence, it was expected to achieve similar results for single applications and applications running in parallel. To prove this an on-line test was done. The test consisted of reading a PDF file while listening to an audio stream. The results listed in Table 6.5 approved the assumption.

6.4 Off-line Evaluation

Because on-line tests are very time consuming, there has to be a possibility to do off-line testing, too. This is especially important for fine tuning of parameters.

Off-line testing is also useful as it can ease the comparison of different approaches, because it allows feeding the different tests with the very same data. Therefore, the data collected from the kernel during the live tests was also logged for off-line testing. In the following section, the approach of single resource profiles presented in Section 6.3 is compared to application profiles introduced in the next section.

6.4.1 Application Profiles

A second approach was to classify application profiles instead of separate resource profiles. This means the classification was no longer done for each resource separately, which would result in three classifications for each feature vector as done in the first approach. Instead only one classification for each feature vector was done, resulting in a class that represents one power management setting composed of the appropriate CPU speed, beacon period and standby time-out. A comparison of both approaches is shown in Figure 6.1. The first approach of single resource profiles is illustrated in the upper half of Figure 6.1, while the lower half shows the second approach of application profiles.

To discover which approach is more reliable, an off-line evaluation of both approaches was done. To simulate the conditions of an on-line test, power management settings were only switched if the classification yields a constant result over five successive classifications. Due to the startup time of the applications, the first fifteen classifications were ignored for computing the rates of correct classifications.

The evaluation was done as follows: To get a single classification rate for the three separate classifications obtained by using resource profiles, a classification result has been defined as correct if the classifications for all three resources were correct; in all other cases, the classification result is defined as false. Then the classification demons of both approaches were fed with the trace files of the on-line test. Table 6.6 lists the rates of correct classifications for each application. The resource profiles yield better results than the application profiles except the video stream test.

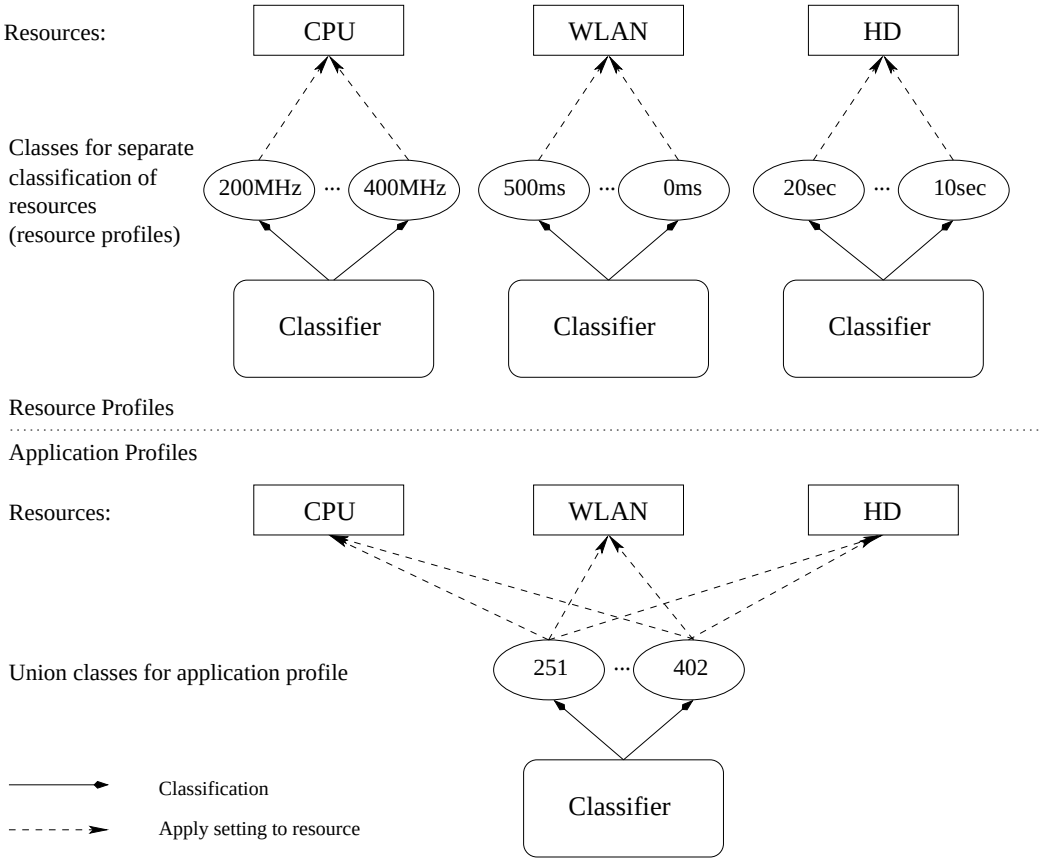

Figure 6.1: Comparison between resource profiles and application profiles

A possible explanation of the worse results of application profiles could be the following:

For both approaches the same potential features were used. The CART has the task to find class borders in feature space. This is rather easy for only two or three different classes but much more complicated for a multiple of classes. This is because for a lot of classes it is more probable that the classes are not separable in feature space.

A consequence of this problem is that especially similar classes such as the four application profiles of the opieplayer can not be distinguished by the classifier. This leads to alternating decisions which result in bad classification rates.

The results of the resource profiles in Table 6.6 and of the on-line test in Table 6.4 differ slightly. This can be explained by the off-line evaluation. Instead of getting the kernel data in real-time, the classification demon receives it all at

Application	Application Profiles	Resource Profiles
konqueror	100%	100%
opieplayer		
• audio from file	95.8%	96.7%
• video from file	49.3%	98.8%
• audio stream	85.8%	92.9%
• video stream	80.7%	72.3%
qpdf	100%	100%
ssh	100%	100%
scp	99.3%	99.3%
showimg	78.8%	100%
patience	90.2%	98.0%

Table 6.6: Comparison between the rate of correct classification of application profiles and resource profiles

once from the trace files. Hence, the evaluation of the results with time stamps is not appropriate. Instead a logic time was used, which was represented by the line numbers of the trace files. Due to scheduling effects, there is a non linear mapping between logic time and real time stamps of the on-line test, resulting in slight differences in the computed classification rates.

6.4.2 Comparison of Different Feature Sets

Until now, the CART algorithm had the possibility to use all available features to build the classification tree. It was also already mentioned that the implicit feature selection done by CART may not be optimal (see Section 3.2). Therefore, the impact of different feature sets should be investigated. As the search for the optimal feature set is very time-consuming, this work focuses on the examination of a few interesting feature sets.

The preceding tests were done on a selection (done by the CART) out of all features. It was supposed to be interesting to investigate the impact of features on the classification results that are calculated on parameters of a resource that differs from the resource the CART is built for. Therefore, the set of all features was split into two disjoint subsets for each resource. The first subset contained all

Application	All features	Related features	Unrelated features
	CPU	**CPU**	**CPU**
konqueror	100%	100%	36.7%
opieplayer			
• audio from file	96.9%	100%	97.6%
• video from file	98.8%	100%	98.8%
• audio stream	99.5%	88.5%	99.4%
• video stream	71.8%	76.2%	97.8%
qpdf	100%	97.8%	100%
ssh	100%	20.8%	100%
scp	100%	100%	100%
showimg	100%	99.1%	100%
patience	97.7%	41.4%	100%

Table 6.7: Resource CPU: Rates of correct classifications for different feature sets

features of parameters that are *related* to the investigated resource. The second subset contains all other, *unrelated* features that are retrieved of the remaining resources. For example, the first subset for the resource CPU contains features like “CPU usage” and “active time” while the second subset includes features as “Number of received packets” or “Number of written sectors of the hard disk”.

First, the classifiers for the resources were trained with the set of *related* features. Then the trace files recorded during the on-line test were used to evaluate this feature set off-line as described in the last section. After that, the same was done for the set of *unrelated* features. The results for the resources CPU, wireless network interface and hard disk are shown in Table 6.7, 6.8 and 6.9, respectively.

According to the results of Table 6.7, the features related to resource CPU appear to be insufficient for reliable CPU speed classification. Especially the classification rates for ssh and patience are poor.

A matter for the very poor result of ssh for the *related* feature set could be that the major computing effort is done on the remote machine whereas on the local machine only displaying is done so that there is only seldom and little CPU usage. Because of that sparse data, the training was not able to find the correct class borders. Instead, perfect classification is possible with the *unrelated* feature set. This is proposed to be due to the characteristic network traffic of a ssh session.

Application	All features	Related features	Unrelated features
	WLAN	**WLAN**	**WLAN**
konqueror	99.7%	100%	22.0%
opieplayer			
• audio from file	—	—	—
• video from file	—	—	—
• audio stream	93.1%	99.2%	83.1%
• video stream	100%	100%	95.5%
qpdf	—	—	—
ssh	100%	100%	100%
scp	99.3%	99.2%	90.0%
showimg	—	—	—
patience	—	—	—

Table 6.8: Resource WLAN: Rates of correct classifications for different feature sets

Patience uses the CPU unsteadily. This is caused by the great amount of user interaction on the one hand and the animation of the cards on the other hand. So there are periods of user thinking which result in unused CPU and periods of animation which is CPU intensive. Thus, the classification decisions alternate which results in a poor classification rate. As another test approved, rising the amount of equal successive classifications to ten would improve the classification rate with *related* features of the resource CPU for patience to about 95%.

The *unrelated* feature set appears to be even better than the *related* feature set for the resource CPU. This is supposed to be an effect of application selection. It was intended to test applications that differ in their resource usage. So as the tested applications differ especially in their usage of the hard disk and/or wireless network interface, this information is suited to identify the applications and/or their CPU performance demands.

The results of Table 6.8 show that the *related* features of the wireless network interface achieve slightly better results than the set of all features. This enforces the suggestion to do feature selection to further improve the achieved results.

The web browser konqueror achieved very poor classification rates for the set of unrelated features of both resources. This is supposed to be because of his

Application	All features	Related features	Unrelated features
	HD	HD	HD
konqueror	—	—	—
opieplayer			
• audio from file	98.1%	99.3%	84.0%
• video from file	99.3%	100%	28.9%
• audio stream	—	—	—
• video stream	—	—	—
qpdf	100%	99.8%	100%
ssh	—	—	—
scp	99.0%	99.3%	99.8%
showimg	100%	100%	100%
patience	—	—	—

Table 6.9: Resource Hard Disk: Rates of correct classifications for different feature sets

special usage characteristic of the wireless network interface. As konqueror is assigned to a beacon period on its own but yields a common CPU usage characteristic for GUI applications, the CART could not correctly determine the beacon period from the CPU usage. Otherwise the special network characteristic could be treated as erroneous at the training for the resource CPU because the konqueror is only one of many applications with a CPU speed of 300MHz.

The *related* features even improve the classification rates slightly for the hard disk as can be seen in Table 6.9.

With exception of few cases for the resource CPU, the features that are related to the resource to classify appear to be most dedicated for classification of that resource. But due to the even rather good results of the unrelated features, the feature selection should not be limited to the related features. A good mixture/selection of all features should yield the best classification rates.

Chapter 7

Future Work

As adaptive power management has not yet received much attention as a research subject, there is a wide range of improvements and further investigations.

To classify the characteristic usage data, features are calculated on that data. A feature should have a value as similar as possible for data according to the same class and should vary as much as possible for data of different classes. It has been shown that a great amount of features can be calculated from the obtained usage data. But more important than quantity is the quality of the features. The Classification And Regression Tree algorithm already does feature rating but the quality depends on the current training set (see Section 3.2). Other training sets may result in a completely different rating. Therefore, further research is desirable to find out which features are best and why, independent from the current training set. This will help to create new and better features and also to reduce the amount of features consulted for classification.

Currently the training of the classifier is done off-line. It would be more comfortable if also the training could be done on-line. This would allow to interactively change the power management preferences of applications or to adapt them to new applications. To achieve this, the training algorithm would have to be ported to the iPAQ. An additional program that offers the possibility to set or change the power management preferences for applications on-line could improve usability for the end user. A change of the power management preferences could then initiate the training algorithm to do new training.

Another task to be evaluated is the scalability concerning the number of ap-

plications running in parallel. In Section 6.3.2 it has been argued that due to accounting with resource containers, the classification rate of applications running in parallel is not reduced. But resource containers only assure separate logging of the usage characteristics. Indeed, the usage characteristics itself could change if there were several concurrent applications. The existence of this effect and its influence on the classification rate has to be analyzed. If it exists, features have to be found that are resistant to that effect.

It is assumed that the adaption of this approach to server environments is also possible. In this context, information about the client/server relations is supposed to be very useful for identifying the performance demands. Using resource containers, client/server relations are automatically identified. So the adaption to that environment should not be complex. Of course, this information can also be used to identify the relations between GUI applications and the X-server.

A more advanced and interesting extension to the developed approach would be to not only adapt the power management mode of the hardware to the appropriate setting, but to adapt the hardware itself to the specific requirements of the currently running application. In combination with reconfigurable hardware like FPGAs, the presented approach would permit to use the optimal hardware configuration for each application.

Chapter 8

Conclusion

This work presents an adaptive power management method to reduce the energy consumption without causing performance loss experienced by the user.

It is argued that the power/performance trade-off resulting from power management is application and user-specific. Therefore, it is necessary to adapt the power management decision to the currently running applications. For this purpose, different usage profiles which represent the user-defined power/performance trade-off are identified.

The power management decision is achieved by considering characteristic usage data which is retrieved from the resources. This information is captured for each application separately on operating system level by the abstraction of resource containers.

A classifier trained by the Classification And Regression Tree training algorithm is automatically integrated into the classification demon. This demon receives the characteristic usage data and computes the according usage profile. Then the appropriate power management settings for that profile are activated.

The approach was evaluated on an iPAQ running a modified Linux kernel and the user space classification demon. An average classification rate of 98% was achieved for several typical mobile applications. Due to the use of resource containers for separate accounting, the classification rate is not reduced for applications running in parallel either.

List of Figures

List of Tables

Bibliography

[1] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation OSDI'99*, February 1999.

[2] Davide Bertozzi, Anand Raghunathan, Luca Benini, and Srivaths Ravi. Transport protocol optimization for energy efficient wireless embedded systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, March 2003.

[3] L. Breiman. *Classification and Regression Trees*. Wadsworth, Belmont CA, 1984.

[4] Maria Calzarossa, Luisa Massari, and Daniele Tessera. Workload characterization issues and methodologies. In Günter Haring, Christoph Lindemann, and Martin Reiser, editors, *Performance Evaluation: Origins and Directions*, pages 459–482. Springer-Verlag, 2000. Lect. Notes Comput. Sci. vol. 1769.

[5] Surendar Chandra. Wireless network interface energy consumption implications of popular streaming formats. In Martin Kienzle and Prashant Shenoy, editors, *Multimedia Computing and Networking (MMCN'02)*, volume 4673, pages 85–99, San Jose, CA, January 2002. SPIE - The International Society of Optical Engineering.

[6] Surendar Chandra and Amin Vahdat. Application-specific network management for energy-aware streaming of popular multimedia format. In *Proceedings of the 2002 USENIX Annual Technical Conference*, June 2002.

[7] IEEE Computer Society LAN MAN Standards Committee. *IEEE 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications*, August 1999.

[8] Compaq, Intel, Microsoft, Phoenix, and Toschiba. ACPI: Advanced configuration and power interface. *http://www.acpi.info*, 1996.

[9] Matthias Faerber. Application-specific power management for wireless networks. Master's thesis, University of Erlangen-Nuremberg, January 2004.

[10] Krisztin Flautner and Trevor Mudge. Vertigo: Automatic performance-setting for linux. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation OSDI'2002*, December 2002.

[11] S. Gelfand, C. Ravishankar, and E. Delp. An Iterative Growing and Pruning Algorithm for Classification Tree Design. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13:302–320, 1991.

[12] K. Govil, E. Chan, and H. Wassermann. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proceedings of the first Conference on Mobile Computing and Networking MOBICOM'95*, March 1995. also as technical report TR-95-017, ICSI Berkeley, Apr. 1995.

[13] Alexander Guy, Jamey Hicks, Russ Nelson, Carl Worth, Ken Causey, Phil Blundell, Jim Gettys, and Koen Kooi. Handhelds.org - the familiar project. *http://familiar.handhelds.org*, 12.30.2004.

[14] S. King, A. W. Black, P. Taylor, R. Caley, and R. Clark. Edinburgh speech tools library. *http://www.cstr.ed.ac.uk/projects/speech_tools*, 1.10.2005.

[15] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the Eighth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM 2002)*, September 2002.

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

[17] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi. Battery-driven system design: A new frontier in low power design. In *Proceedings of the 7th Asia and South Pasific Design Automation Conference and 15th International Conference on VLSI Design (VLSI Design / ASPDAC'02*, January 2002.

[18] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Requester-aware power reduction. In *International Symposium on System Synthesis*, pages 18–23. Stanford University, September 2000.

[19] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Power-aware operating systems for interactive systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(2), April 2002.

[20] Yung-Hsiang Lu and Giovanni De Micheli. Adaptive hard disk power management on personal computers. In *Proceedings of the IEEE Great Lakes Symposium*, pages 50–53, March 1999.

[21] David M. Magerman. *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University, 1994.

[22] M. Neufeld, D. Grunwald, P. Levis, C. Morrey, and K. Farkas. Policies for dynamic clock scheduling. In *Proceedings of the Forth Symposium on Operating System Design and Implementation OSDI'2000*, October 2000.

[23] H. Niemann. *Klassifikation von Mustern*. Springer–Verlag, Berlin, 1983.

[24] Odysseas I. Pentakalos, Daniel A. Menasc, and Yelena Yesha. Automated clustering-based workload characterization. In *5th NASA Goddard Mass Storage Systems and Technologies Conference*, September 1996.

[25] T. Pering and R. Broderson. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'98*, June 1998.

[26] Martin Waitz. Accounting and control of power consumption in energy-aware operating systems. Master's thesis, Department of Computer Science 4, January 2003. SA-I4-2002-14.

[27] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the First Symposium on Operating System Design and Implementation OSDI'94*, November 1994.

[28] Andreas Weissel, Bjoern Beutel, and Frank 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.

[29] Andreas Weissel, Matthias Faerber, and Frank Bellosa. Application characterization for wireless network power management. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS'2004)*, January 2004.

[30] Heng Zeng, Xiaobo Fan, Carla Ellis, Alvin Lebeck, and Amin Vahdat. Ecosystem: Managing energy as a first class operating system resource. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'02*, October 2002.

Anwendungsspezifische Energieverwaltung in Betriebssystemen

In den letzten Jahren ist die Nachfrage nach mobilen Geräten stark gestiegen. Da diese auf die begrenzte Energiereserve von Batterien angewiesen sind, ist ein sparsamer Umgang mit Energie unerlässlich. In der Forschung gilt deshalb Energieverwaltung sowohl für mobile Geräte als auch für eingebettete Systeme und Allzweckbetriebssysteme als wichtiges Aufgabengebiet. Es gibt bereits eine Vielzahl von Verfahren, die den Energieverbrauch von einzelnen Komponenten, aber auch vom gesamten System verringern. Oft haben die bekannten Verfahren allerdings den entscheidenden Nachteil, dass sie nicht nur den Energieverbrauch, sondern auch die vom Benutzer wahrgenommene Leistung des Systems reduzieren.

Der in dieser Arbeit vorgestellte Ansatz geht davon aus, dass es sinnvoll ist, nur dann Energie zu sparen, wenn die Leistung der Anwendungen von den Energieeinsparungen nicht negativ beeinflusst wird. Dazu wird ein System vorgestellt, das dem Benutzer erlaubt, die seinen Leistungsansprüchen genügenden Energiesparmodi für Anwendungen zu spezifizieren. Diese Einstellungen werden dann im Betrieb dynamisch vorgenommen. Um in den für eine Anwendung spezifizierten Energiesparmodus zu wechseln, muss das Verfahren erkennen welche Anwendung gerade verwendet wird. Dazu könnte man einfach den Energiesparmodus mit dem Namen der Anwendung verknüpfen. So wäre es allerdings nicht möglich einer Anwendung verschiedene Energiesparmodi zuzuweisen. Mehrere Energiesparmodi sind aber für Anwendungen die abhängig von der Nutzung sehr unterschiedliche Leistungsanforderungen haben können sinnvoll. Zum Beispiel belastet eine Anwendung zum Abspielen von Multimediainhalten den Prozessor

sehr unterschiedlich, je nachdem ob eine Audio- oder Videodatei abgespielt wird. Deshalb werden die Energiesparmodi nicht den Anwendungen sondern sog. Nutzungsprofilen zugeordnet. Da aber keine direkte Information darüber vorhanden ist in welcher Weise die laufende Anwendung verwendet wird, wird versucht das Nutzungsprofil der aktuellen Anwendung an Hand von Nutzungscharakteristika zu erkennen. Unter Nutzungscharakteristika versteht man statistische Daten über die Verwendung von Betriebsmitteln.

Für das Sammeln der Nutzungscharakteristika werden „Resource Container“ verwendet. Diese ermöglichen die Abrechnung von Dienstleistungen des Betriebssystems unabhängig von der Prozessstruktur. Durch eine Erweiterung der Resource Container ist es zusätzlich möglich die Verwendung des Prozessors, die Sende- und Empfangscharakteristik der drahtlosen Netzwerkkarte und die Zugriffsmuster der Festplatte zu protokollieren.

Um die laufende Anwendung zu erkennen werden Methoden aus dem Bereich der Mustererkennung verwendet. Die Nutzungscharakteristika werden mit Hilfe eines Klassifikators auf die Nutzungsprofile abgebildet. Als Klassifikator werden Entscheidungsbäume verwendet. Diese ermöglichen die Zuordnung von Eingabedaten zu Klassen indem sie sukzessiv binäre Entscheidungen treffen. Vorher müssen die Entscheidungsbäume trainiert werden. Dazu muss eine klassifizierte Stichprobe vorliegen, d. h. der Benutzer muss in der Trainingsphase den Trainingsdaten den gewünschten Energiesparmodus zuordnen. Dann wird die Trainingsmenge mit Hilfe des Entscheidungsbaums so geteilt, dass die den Blättern des Baums zugeordneten Teilmengen möglichst homogen bezüglich ihrer Klassenzugehörigkeit sind. Der entstandene Entscheidungsbaum dient als Klassifikator für die Nutzungscharakteristika.

Im laufenden Betrieb werden zunächst die Daten der Betriebsmittelnutzung mit Hilfe der Resource Container protokolliert. Diese Daten werden regelmäßig abgeholt und an den Klassifikationsprozess weitergeleitet. Dieser berechnet auf den empfangenen Daten Merkmale, die zur Klassifikation mit dem Entscheidungsbaum verwendet werden. Die bei der Klassifikation erkannten Energiesparmodi werden anschließend mit bereits vom Betriebssystem bereitgestellten Hilfsprogrammen aktiviert.

Dieses System wurde für einen iPAQ mit einem PXA250 Prozessor, der

Veränderungen der Taktrate und der Betriebsspannung ermöglicht, einer drahtlosen Netzwerkkarte und einer Festplatte implementiert. Mehrere für mobile Geräte typische Anwendungen wurden zur Evaluation des Ansatzes untersucht.

Nahezu alle Anwendungen wurden in einem Test unter realen Bedingungen richtig erkannt, wodurch eine durchschnittliche Erkennungsrate von ca. 98% erreicht wurde. Dank der separaten Protokollierung der Nutzungsdaten jeder Anwendung durch die Resource Container kann diese Erkennungsrate auch erreicht werden, wenn mehrere Anwendungen gleichzeitig ausgeführt werden. Um Konflikte zwischen den spezifizierten Energiesparmodi parallel laufender Anwendungen aufzulösen wird eine einfache Strategie genutzt: es wird immer der Energiesparmodus verwendet, der die größere Leistung bietet, aber auch geringere Energieeinsparungen ermöglicht. Allerdings ist es so möglich, dass auch die Leistungsansprüche der leistungshungrigeren Anwendung erfüllt werden.

Es wurde versucht, an Stelle von einzelnen Energiesparmodi für jedes Gerät einen globalen Energiesparmodus für alle Geräte zu klassifizieren. Dies hat zu keiner Verbesserung der Ergebnisse geführt. Weiterhin wurde der Einfluss von verschiedenen Merkmalsmengen auf das Klassifikationsergebnis untersucht. Hierbei hat sich gezeigt, dass die Merkmale, die von dem zu klassifizierenden Betriebsmittel stammen, oft aber nicht immer ausreichen, um sehr gute Klassifikationsergebnisse zu erzielen. Da allein mit Merkmalen von anderen Betriebsmittel auch gute Ergebnisse erzielt werden konnten, erscheint eine Auswahl aus beiden Merkmalsarten als die beste Lösung.