ABSTRACT

(Computer Science)

EXPLICIT ENERGY RESOURCE MANAGEMENT AS A FIRST CLASS OPERATING SYSTEM RESOURCE

by

Heng Zeng

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Carla S. Ellis, Supervisor

_____
Alvin Lebeck, Supervisor

_____
Daniel Sorin

_____
Amin Vahdat

_____
Jun Yang

An abstract of a dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2004

EXPLICIT ENERGY RESOURCE MANAGEMENT AS A FIRST CLASS OPERATING SYSTEM RESOURCE

by

Heng Zeng

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Carla S. Ellis, Supervisor

_____
Alvin Lebeck, Supervisor

_____
Daniel Sorin

_____
Amin Vahdat

_____
Jun Yang

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2004

Abstract

Energy consumption has recently been widely recognized as a major challenge for computer system design, especially for mobile systems. The energy constraint will often impact the performance of devices as well as applications. To address this problem, previous works focus on improving energy efficiency of hardwares or reducing energy demand of applications. This thesis advocates a new way to look at the energy problem. It is argued that energy should be managed explicitly as a first-class system resource by the OS and energy-saving efforts across the system should be coordinated in order to achieve global energy-related goals.

This thesis proposes the $currentcy$ model as the basic abstraction for the energy resource. It represents an application's right to consume a certain amount of energy within a fixed amount of time. Currentcy is allocated to applications according to their importance and can be spent on any hardware device to pay for the energy consumption of their activities. Through the management of currentcy, including the currentcy allocation to applications, the currentcy consumption scheduling on devices and currentcy payback, energy could be managed explicitly. A currentcy based energy management framework is proposed in this thesis and it is demonstrated that the battery lifetime (and thus the system lifetime) can be extended while the execution of critical applications can be guaranteed.

The energy resource is a special resource with many unique characteristics. It has impact on all applications as well as all devices, and its management is an unified effort from all system components. These characteristics directly affect the design of the management framework and the management policies. In this thesis, an embedded energy model approach is proposed for energy accounting, the currentcy conserving allocation policy is proposed for efficient management and the currentcy centric scheduling policy is proposed to provide prioritized service to applications. Working within the management framework,

these policies can enforce the global goals without application cooperation.

In addition to global management and energy quota enforcement, the management framework is also capable of assisting various local energy-saving efforts. A currentcy-based hard disk scheduler is introduced that schedules hard disk accesses by pseudo-economic activities such as bidding, pricing and debiting. It is shown that by exposing currentcy information to the hard disk, energy can be saved by smarter decisions without violating the global goals. The application-OS interaction for better energy management is also discussed in this thesis.

A prototype system, ECOSystem, is implemented based on the Linux kernel on a laptop platform. The currentcy model, the management framework and the management policies are evaluated using a variety of synthetic benchmarks and real applications. Experiment results show the success of the explicit energy management approach in achieving global energy-related goals and in coordinating various energy-saving efforts.

Acknowledgements

First of all, I'd like to thank my advisors, Alvin Lebeck and Carla Ellis, for their guidance for my development, their advising me on my research, their encouragement and not to mention their financial support. They provided many valuable insights and comments for my research and we worked very closely on my published papers which constitute the main body of this dissertation. They are not only great advisors to me, their intelligence, diligence and kindness also motivated me during my years at Duke university.

Thanks also to other people that provided critical feedback and advice to this dissertation. Thanks to Amin Vahdat for his quick feedback during our weekly Milliwatt meeting. Thanks to Daniel Sorin for his comments on my prelim and his suggestion of control theory analysis. I would also like to express my gratitude to my office mates, Xiaobo Fan and Tong Li, for their comments on my thesis and frequent discussions on my research. Special thanks to Dianne Riggs, she is always ready to help out with all the paper works and always kindly remind me of before the deadline. In addition, I am grateful to all members of the Milliwatt project, who helped me develop my ideas and improve my presentation skill.

During my years as Ph.D. candidate, I am deeply indebted to my wife, Wenshu. I cant imagine I can finish my Ph.D without her support and sacrifice. I am also deeply grateful to my father for his impartment of scientific thinking during my very early years. Thanks to my mother for her care and believe in me. Both my parents made great sacrifices to finance my education to ensure that I received the best education possible. I am extremely happy to be in such a wonderful family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Energy has became one of the most critical resources in a mobile system. Both the design of hardware and the choice of applications are greatly limited by the energy available in the battery. Attacking the energy bottleneck is a system wide effort. There already have been many works on energy saving techniques targeting various components of the system. This thesis, however, argues that attacking the energy bottleneck should also be a unified effort. All energy consumptions come from the same pool. The energy consumed by one hardware unit can not be consumed by another one, and energy saved by one application can be used to power the activity of another application. This thesis advocates that energy should be managed explicitly as a first-class system resource. Energy management at different system components should be unified and the user should be given a goal-oriented control over the system. Instead of directly adjusting the energy management of every device, the user can simply specify some goals to be achieved, such as the system to last a certain amount of time or guaranteeing the running of a specific application. Only with explicit energy management, can energy consumption across the system be coordinated towards these global goals.

This chapter presents a high-level synopsis of the explicit energy management approach. Next, section 1 overviews previous energy saving efforts. Then section 2 briefly discusses the special characteristics of energy and how it should be managed. This is followed by highlights of the main contributions of the thesis, and finally the organization for the rest of the thesis.

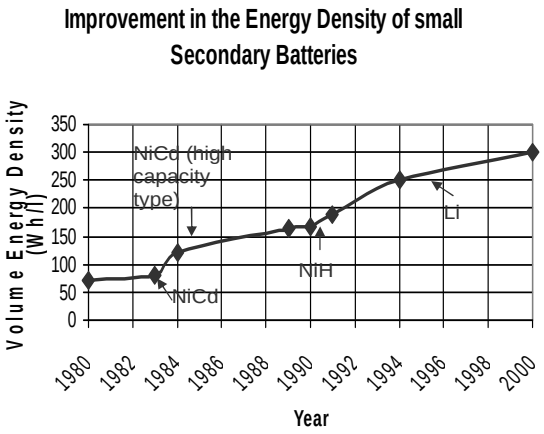


Figure 1.1: Battery Density Improvement in Recent Years

1.1 Attacking Energy Bottleneck

Today, mobile devices are becoming increasingly popular, from laptops, PADs, cell phones to emerging platforms such as wireless sensor networks. In these environments, available energy, as embodied by the system battery, plays an increasingly important role. As shown in figure 1.1, the battery energy density has doubled every ten years with the introduction of new battery technology. But at the same time, the CPU's power consumption has increased by a factor of ten in the last decade. The battery bottleneck is exacerbated by the demand to miniaturize these devices yet provide prolonged battery lifetime. Energy has became one of the most critical resources for mobile system; the computational power that can be provided by a mobile device is often not limited by the raw speed of the hardware, but by the amount of energy in the battery.

1.1.1 Existing Piece-wise Efforts

Ideally, the energy problem should be addressed at all levels in the system. As shown in figure 1.2, currently the energy problem is mainly addressed at the hardware level and the application level with a piece-wise approach. There has been relatively little attention to

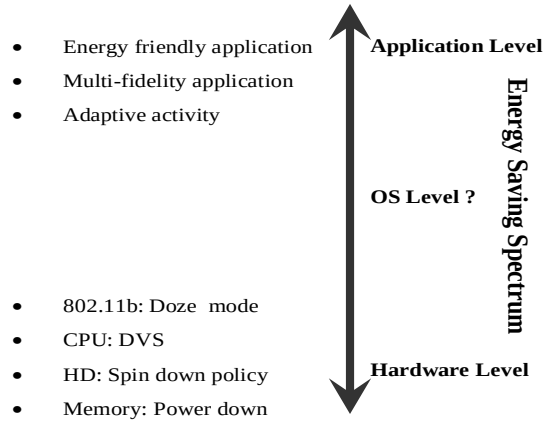


Figure 1.2: Energy Saving Spectrum

treat energy as a first-class system resource and manage it explicitly.

At the hardware level, energy consumption can be reduced at each device by low-power circuit design and architecture, such as reduced voltage and clock gating [10, 12, 49]. For mobile systems, the metric to measure the hardware performance has changed from MIPS to MIPS/J(MIPS per Joule). In addition to low-power design, hardware devices can provide device-specific power management features, which can then be exploited by power management policies at higher level of the system [33, 25]. For instance, with the voltage scaling technique, a CPU can provide a series of performance modes with increased speed but with decreased MIPS/J. A voltage scaling policy monitors the computation demand from application and chooses the performance mode that meets the application's demand with the highest MIPS/J. Similarly, ATA standard for hard disk interface defines a set of power modes that can be controlled by the OS. The 802.11b standard for wireless network interface incorporates a doze mode that allows the device to be turned off when there is no network traffic.

At the application level, energy can be saved by modifying applications to be energy aware. For instance an energy friendly application can give hints to the device manager or even change its device access pattern to increase the energy efficiency [58]. An application

can also adapt to the energy constraint with reduced application activity [22, 21, 43].

Without a framework for explicit energy management, existing piece-wise efforts at the hardware level focus on saving energy with limited impact on performance [33, 11, 18]. They fail to address the opposite scenario where energy is the most critical resource and the overall performance should be adjusted to achieve energy-related goals. At the application level, the underlying assumption of application adaptation is that reduced application activity will result in reduced energy consumption. However this assumption is not always true without hardware cooperation. For example without unified management, the hard disk may not be aware of the reduced number of accesses from the application and consume about the same amount of energy by keeping the disk spinning. Another problem with this approach is that without accurate energy accounting, the adaptation is problematic especially when there are multiple applications and asynchronous activities. The application adaptation approach also relies heavily on voluntary performance degradation, energy related goals can not be achieved with the presence of traditional energy oblivious applications.

1.1.2 Energy Management in the OS

Imagine a scenario of using a laptop during a flight and relying on the battery as the only power source. The environment is highly dynamic because the application may change its power requirements and there may be several applications running concurrently (word and MP3 player for example). Instead of directly adjusting the power mode of each device and the performance level of every application, the user should be able to specify some global goals to be achieved. For instance, the user might simply specify that the battery has to last throughout the flight and the energy requirement of the most important application has to be met. Similar scenarios also exist in other platforms, for instance a PDA may have to function continuously during hiking, or a wireless sensor node may have to guarantee the

energy requirement of a certain task during the night before being re-charged by the sun-light. While previous energy-saving efforts are critical to alleviate the energy bottleneck, their piece-wise approach can not guarantee these global goals.

This thesis advocates that energy should be managed explicitly as a first-class system resource by the OS and the OS should be the central point for the unified effort of efficient energy management. By managing energy explicitly, energy related-goals can be met and energy supply to tasks can be prioritized according to their importance. Previous piece-wise energy-saving efforts can also be incorporated into the unified energy management with a global view.

1.2 Managing Energy as a First Class System Resource

Attacking the energy bottleneck should be a system wide unified effort. The OS is the only entity with a global view and it is argued in the previous section that energy should be managed explicitly by the OS as a system resource. However, there are several questions still to be answered. The first question is what are the specific goals to be achieved by explicit energy management. There are many system resources managed by the OS, such as the memory, CPU, semaphore, the open file descriptor etc and they have been studied extensively. The second question is what is special about the energy resource that requires extra research. The last question is how energy management is different from the management of other resources and how these system level goals can be achieved. These questions are discussed in the rest of this section.

1.2.1 Goals of Energy Management

Just as managing other system resources, the management of the energy resource involves several responsibilities. First of all the resource utilization must be planned. Secondly the limited resource must be shared by applications and resource quota be enforced to prevent

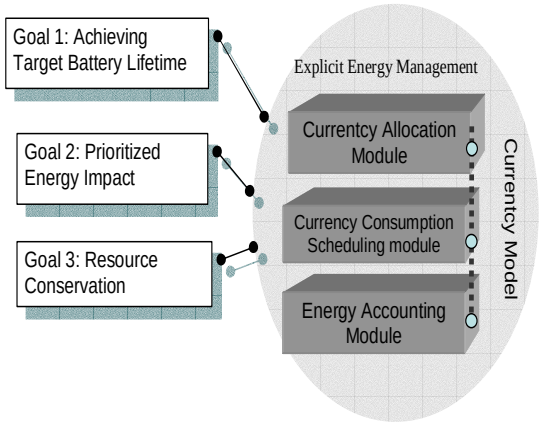


Figure 1.3: The Goals and Modules of Energy Management

a "run-away" process harming other applications. Finally the resource must be managed efficiently. For instance idled resource can be reclaimed. The resource manager could also interact with the application to meet its special demands as long as the resource quota is not violated.

To be more specific, the goals that orient the research in this thesis are shown in figure 1.3. The first goal is achieving target battery lifetime. This is because energy becomes a limited resource when an extended target lifetime is specified. The second goal is to distribute the performance impact among applications by resource sharing. By guaranteeing the energy supply to the most critical application, the user's preference can be satisfied. The third goal is resource conservation for efficient energy management.

One assumption of the energy management of this thesis is that the there is no modification in the application to be energy aware. While this work will certainly benefit from application cooperation, application rewriting is very expensive for general purpose applications and we'd like to see what can be achieved by the OS alone. In chapter 8, there are some discussions on how OS can interact with devices and applications for better energy management.

1.2.2 Characteristics of Energy Resource

Energy has many special characteristics that affect its management. Some of them are listed below.

Global impact on all devices and applications. All tasks and devices need energy to run. This characteristic argues for a unified model to quantify the energy resource and the performance impact on all tasks and devices.

Two-dimensional distribution. The energy resource is first allocated to tasks and then consumed by devices. Each task may activate multiple devices concurrently and each device may serve several tasks at the same time. A mechanism is needed to bridge the energy allocation to tasks and the energy consumption by devices.

Co-existence with other resources. Energy is not an independent resource. It can only be consumed with the activity of other devices. Support from hardware devices is essential for successful energy management.

Even though the energy resource has many special characteristics, its management shares the same responsibilities as other traditional resources. The effort in this thesis is not trying to develop a totally new approach for energy management. Instead this thesis studies the basic concepts behind existing resource management approaches and sees how they can be applied to the resource of energy.

1.2.3 Currentcy Model and Management Framework

The first step of this thesis towards explicit energy management is the proposal of the $currentcy$ model as an abstraction to quantify the energy resource. Currentcy represents the right to consume some amount of energy for an application. Energy resource is allocated to applications in the form of a currentcy credit which can be spent on any device.

Currentcy is deducted from a task as energy is consumed and the OS will deny any service if the task has no more currentcy left. Currentcy is not only the foundation for implementing energy management modules, it is also the abstraction for unifying device management policies and for the interaction between the OS and applications.

Based on the currentcy model, an energy management framework is constructed around the flow of currentcy. Because of the distributed nature of the energy resource, its management is realized by the cooperation of three separate function modules,the accounting module, the allocation module and the scheduling module, as illustrated in figure 1.3. An accurate accounting module is the prerequisite of energy management. The currentcy allocation module plans the energy consumption among applications and the scheduling module schedules the application on devices to actually consume the currentcy. These functions are usually integrated into one algorithm for many traditional resources. For the energy resource, they are distributed across the system and are coordinated by the currentcy model. The currentcy model and the management framework are discussed in detail in the next chapter.

1.3 Contributions

The main point of this thesis is that energy should be the first-class system resource explicitly managed by the OS to unify energy saving efforts across the system. To validate this, this thesis explores across a range of areas essential for energy resource management, including the management framework, the function modules and their interaction, the policy space and the device and application interface. To be more specific, our contributions include:

1. *Currentcy model and management framework.* An unified model, Currentcy, is proposed as the basic building block for explicit energy management. It flows across

the system to unify various components of the system. The energy management framework of this thesis can achieve the proposed goals through the management of currentcy.

2. *ECOSystem prototype.* To validate the currentcy based management framework, a prototype system, called ECOSystem, is implemented on a laptop platform. ECOsystem is also used as a test-bed to evaluate energy management policies and OS interface to applications and devices.

3. *Embedded energy model approach for energy accounting.* Accounting for energy resource is difficult because many energy consumption activities are asynchronous. An embedded energy model approach is used to track the applications responsible and attribute the energy consumption correctly to each application.

4. *Currentcy conserving allocation.* The goal of currentcy allocation is to share the limited currentcy among tasks. At the same time the surplus currentcy from abundant applications is reclaimed by our currentcy conserving allocation and given to other applications in need of currentcy.

5. *Energy centric scheduling.* To guarantee the execution of an important application, the application not only needs enough currentcy , it also needs scheduling opportunities to spend the allotment. In cooperation with currentcy conserving allocation, the energy centric scheduling can provide differentiated service to applications according to their importance.

6. *Self-pacing CPU scheduling.* When the energy resource is limited, the execution of an application has to be throttled. For a CPU without DVS capability, this is realized in ECOSystem by inserting idle CPU quanta into the application's execution thread. With the self-pacing feature, idle quanta are inserted evenly so that the application

can receive steady service rate, which is reflected in many applications as smoothed execution.

7. *Pseudo-economic hard disk access scheduling.* To demonstrate the ability of the energy management framework to unify various energy saving efforts, a hard disk access scheduler is implemented based on pseudo-economic activities such as bidding, pricing and debiting. It is shown that by interacting with the energy management framework, the hard disk access scheduler can make more informed decisions to reduce the energy cost per access.

In addition to those contributions on the energy resource management, this thesis also makes the following contributions:

1. *System stability analysis with control theory.* ECOSystem is a complex system with many coordinated components. It is subjected to many disturbances such as environmental signals or internal errors. In this thesis, it is shown how control theory can be used to analyze the stability of the system and to improve the system's robustness to these disturbances.

2. *Hardware recommendation.* The experiments of this thesis on evaluating the energy management framework also reveal the recommendations for hardware power characteristics. In particular, it is shown that the idle power consumption of a system can greatly affect the management flexibility, and the power state transition cost of a device has a large impact on its energy efficiency.

3. *Discussion of energy management API.* A discussion of application involvement in the OS energy management is included in this thesis. It is shown that it is possible for the energy management framework to better serve an application with some simple cooperations from the application.

1.4 Organization

This chapter has reviewed some background knowledge about energy management and introduced the currentcy based energy management framework. The rest of this thesis is organized as following. The next chapter presents the currentcy model and the management framework in detail. Chapter 3 introduces the implementation of ECOSystem prototype and introduces the applications used for evaluation. Chapter 4 discusses the accounting challenge for the energy resource and proposes the embedded energy model approach. In chapter 5, combining previous work on ECOSystem prototype and energy accounting, the currentcy based energy management framework is evaluated with the goal of achieving target battery lifetime. A feedback module based on control theory analysis is proposed to enforce this goal with the existence of modeling errors. Chapter 6 studies the currentcy conserving allocation policy with the goal of reducing residual battery energy and chapter 7 studies the currentcy centric scheduling policy with the goal of providing differentiated service to applications. ECOSystem's interaction with hardware devices as well as with applications are discussed in chapter 8. Chapter 9 discusses a wide variety of research related to energy management. Finally, chapter 10 concludes this thesis and discusses directions for future research.

1.5 Summary

With the available battery energy being the bottleneck for mobile systems, the energy resource has become one of the most critical system resources. This chapter advocates the approach of explicit energy management to coordinate the energy consumption among hardware devices and to distribute the performance impact to applications. The proposed currentcy model and the currentcy-based energy management framework are briefly introduced in the chapter. This chapter also includes a list of the contributions and the organi-

zation for the rest of this thesis.

Chapter 2

Currentcy Model

The previous chapter shows that the energy resource has some special characteristics. This chapter discusses how these characteristics pose challenges to its management, which leads to the proposal of an unified model and a framework for energy management. The policy space to be explored within the framework is outlined at the end of this chapter.

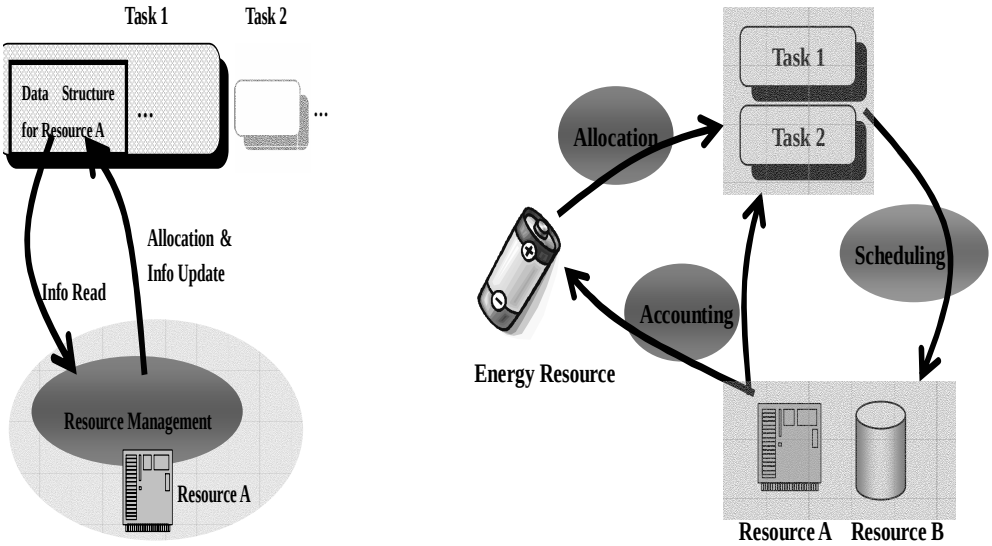
2.1 Energy Management Challenges

With energy becoming the major limiting factor in many computing environments, there have been many efforts in minimizing the degree to which energy becomes the bottleneck. This thesis advocates that energy should also be treated as a first-class system resource, which acknowledges the scenario of limited energy inside the battery that may not be able to satisfy the demand of every application. One of the goals of managing energy as a resource, the same as the goal of managing many other resources in the system, is to distribute the performance impact caused by the bottleneck among all tasks in the system.

Although the managements of conventional resources are studied extensively in literature, managing this new type of resource is still a great challenge because of its unique characteristics described in the previous chapter. The following subsections discuss the specific challenges in more detail.

2.1.1 Distributed Function Modules

The first challenge is that, unlike the management of other resources that usually need only one centralized algorithm, the management of energy can be separated into several



a) Management of Conventional Resources b) Energy Management

Figure 2.1: Resource Management Architecture

function modules distributed across the system. For example, to manage the "conventional" resource of CPU, most operating systems only need one algorithm that selects a process from the waiting queue for the next quantum. However, for the energy resource, the management is further divided into three function modules: the allocation module, the scheduling module and the accounting module.

This difference is caused by the fact that the management of a conventional resource usually involves two parties: the resource and a requesting task set, while the energy management involves three parties: the energy resource, task set and energy consuming resources. Still using the CPU resource as the example, its management consists of a round trip of two steps, as illustrated in figure 2.1-a. In the first step, information, including process priorities and recent CPU usages etc., is read from the processes in the requesting set and the management algorithm decides which process is selected for the next quantum. In the second step, the process is scheduled and resource usage information is updated. With this round-trip process, the resource information is accessed only by the management

algorithm, thus it is common to for it to be kept as private internal data structure, usually meaningless to other parts of the system. In the Linux CPU scheduler, for instance, an integer $count$ is used for each task to indicate both its priority and recent history of CPU usage. In the CPU scheduler of FreeBSD, ready tasks are organized into several queues. A task's priority and CPU usage is reflected in the queue it is in and its position in the queue [42].

In contrast, the management of energy consists of a circle of three steps. As shown in figure 2.1-b, in the fist step, the allocation module allocates energy from the battery to tasks. In the second step, the allocated energy is consumed by devices as tasks are executed by the scheduling module. Here the term $device$ is used for any energy consuming hardware resource. Finally the energy consumed by the tasks needs to accounted and recorded. The remaining energy in the battery should also be updated by the accounting module.

With the distributed manner of the energy management, the challenge is to establish a framework to guide separated function modules towards common goals. To communicate and coordinate different modules, a common model is needed for the energy resource information instead of the private data structures approach adopted in the managements of many conventional resources.

2.1.2 Scheduling of Global Scale

In traditional general purpose systems, resources are managed locally and independently. When a thread of instructions is executed, resources are requested when they are needed. Thus the management policy of one resource only has a partial impact on the overall performance. For instance when the CPU is occupied, a task can still receive, process and buffer network packets at the network interface. The energy resource is different from traditional resources in that it is directly related to the overall performance of an application. When the energy resource is limited, all hardware activities and thus the overall performance of

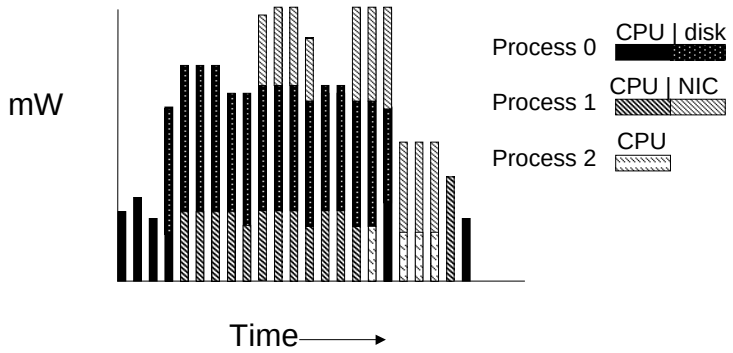


Figure 2.2: Accounting challenges of multiple devices and processes

application will be directly affected.

The management of traditional resources may be coordinated [50, 9], but there is no single resource that unifies the management of all other resources. Another difference of the energy resource is that the scheduling of energy consumption can not be implemented independently. Instead, because energy is ultimately consumed by hardware, its scheduling can only be realized by the coordinated scheduling of all devices. The challenge is to unify the management of diverse local devices towards a common metric of energy consumption.

2.1.3 Accounting for Asynchronous Activities

OS-level energy management can be split across two dimensions. Along one dimension, there are a wide variety of devices in the system (e.g., the CPU, disks, network interfaces, display) that can draw power concurrently and are amenable to very different management techniques. In the other dimension, these devices are shared by multiple applications. The power usage of two simultaneously active hardware components may be caused by two different applications. For example, the disk may be active because of an I/O operation being performed by a "blocked" process while another process occupies the CPU. This presents additional challenges for accounting. Consider the scenario portrayed in Figure 2.2 involving three different processes and three different resources (CPU, disk, and wireless card).

During the highest levels of power consumption, process 0's disk activity, process 1's network usage, and CPU processing by any one of the three processes all contribute. Using a program counter sampling technique, based on time as in PowerScope [22] or energy consumption [13], would inaccurately attribute power costs to the wrong processes.

2.2 Currentcy Model

It is shown in the previous section that the energy resource is not simply another resource to be managed. The characteristics of the energy resource and the management challenges they pose call for a common model and a framework to unify the energy management efforts across the system.

A common unit, $currentcy$, is proposed as the basis for explicit energy management. The created word $Currentcy$ is a combination of $current$ and $currency$, to express the idea that just like money can be used as the unified abstraction of buying power for all kinds of commodities, $currentcy$ is the unified abstraction for energy that can be spent on all kinds of devices. Currentcy is allocated to tasks for gaining accesses to devices. One unit of currentcy represents the right to consume a certain amount of energy within a fixed amount of time.

There are some differences between a unit of currentcy and an equivalent x Joules of energy from the battery. First of all, currentcy is an OS internal abstraction instead of real energy. When currentcy is allocated or charged to a task, the actual energy consumption may not happen at the same time. There is a time limit on the currentcy, meaning that it can be forfeited when it expires. Secondly, currentcy is the model for the purpose of regulating energy consuming activities of managed applications and devices. Only the portion of total energy consumption related to such activities is modeled as currentcy. For example, keeping system time accurate is essential for the running of the whole system. There is no need to regulate this activity and thus no need to model this part of energy consumption.

Currentcy becomes the basis for characterizing the power requirements and gaining access to any of the managed hardware resources. It is the mechanism for establishing a particular level of power consumption and for sharing the available energy among competing tasks. The framework based on the currentcy model for energy management is introduced in the next section.

2.3 Energy Management Framework

This section describes the framework that achieves the energy related goals through the management of currentcy. In the framework, the management of currentcy can be classified into three steps.

2.3.1 The Three-step Approach

The energy management framework is illustrated in figure 2.3. There are three orthogonal dimensions of energy management: time, tasks and devices. In the first step, when a target lifetime is specified, the time is divided into epochs of fixed time length. Each epoch a certain amount of currentcy is generated based on information such as the remaining energy in the battery, target lifetime and platform power characteristics, etc. This is the total amount of currentcy for all tasks during the epoch, and also the amount of currentcy for all devices. In the second step, this limited amount of currentcy is allocated among tasks according to their importance. Tasks can then be scheduled to access devices with the allocated currentcy. In the last step, as energy is consumed by devices, the corresponding amount of currentcy is charged from the tasks. During this step, the currentcy allocated to the tasks is mapped to the currentcy received by the devices. The three steps are further described in the rest of this section.

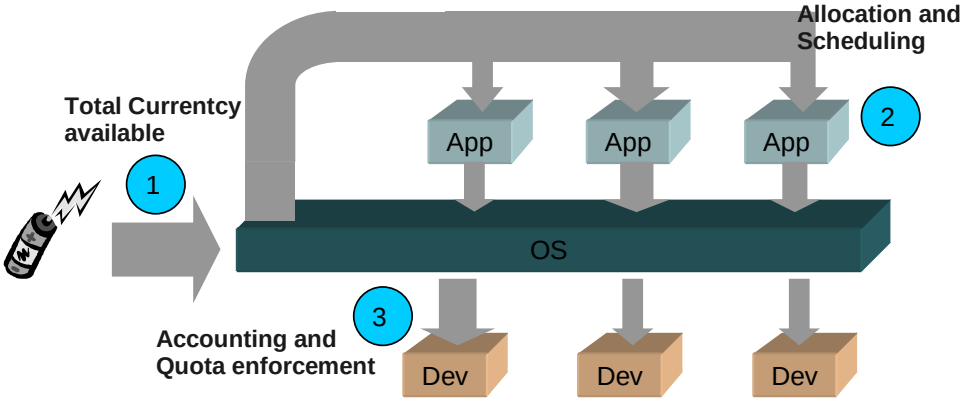


Figure 2.3: Energy Management Framework

2.3.2 Step 1: Energy Management Over Time

Of the three dimensions, energy is first managed along the time dimension for the goal of target lifetime. Currentcy, as the abstraction for energy consuming activities of tasks and devices, is generated in this step.

System Energy Classification

To generate currentcy, the system power costs are characterized into two parts in the framework. The first part is the base energy consumption that includes the low power states of the explicitly energy-managed devices as well as the default state of the devices not yet being considered. The larger the proportion of the system that gets included in the base category, the less opportunity there will be to affect improvements on top of it.

The second part of system energy consumption is the activity power consumption including the active states of the explicitly managed devices. Thus, the halted state of the CPU and the spindown state of the disk fall into the base energy consumption while the active state of CPU and the spinning state of the disk are considered to be in the category of activity power consumption.

The energy management framework regulates battery discharge rate by controlling the

activity power consumption. Currentcy is generated each epoch according to the activity power consumption allowed by the management framework. With this classification, an application can not activate any managed device without currentcy. Also if a device is not accessed and stays in the base power state, no task needs to be responsible for its energy consumption.

Currentcy Distribution Over Target Lifetime

The first level of energy management is to decide how energy is consumed over the target lifetime. This level of energy management is referred to as currentcy distribution over time. At the start of each epoch, the system allocates a specific amount of currentcy available collectively to all tasks in the system. The amount is determined by the discharge rate necessary to achieve the target battery lifetime. By distributing less than 100% of the currentcy required to drive a fully active system during the epoch, components are idled or throttled.

2.3.3 Step 2: Currentcy Sharing

The second step of the framework is the currentcy sharing among competing tasks. Currentcy sharing is accomplished by the cooperation of the allocation module and the scheduling module. The allocation module allocates the total currentcy available each epoch to tasks according to certain criteria that reflect the user's preference. While the currentcy allocated by the allocation module enables a task to access devices, the currentcy consumption scheduling module provides opportunities necessary for the task to spend the currentcy. With the scheduling module, the differentiated resource allocation is transformed into differentiated resource consumption.

The energy management framework allows a task to accumulate surplus currentcy from one epoch to another, enabling a task to pay for one-time large currentcy expenditures.

Without currentcy accumulation, for example, a task that receives 100 currentcy every second can only have a maximum of 100 currentcy in its account. It will be in negative balance by periodic charge of 200 currentcy every 5 seconds even it can afford the charge in average. However, allowing the task to accumulate too much currentcy will cause spiked energy consumption that may be harmful for the battery and may interfere with the execution of other tasks. A task that hoards currentcy unproductively to the end of lifetime increases the residual energy in the battery, which means lost opportunities for other tasks to do more work. The management framework puts a cap on the maximum amount of currentcy can be accumulated by each task. Any currentcy accumulated above the cap is forfeited by the system.

2.3.4 Step 3: Currentcy Payback

In the last step, currentcy is deducted from tasks as they are executed. But locating the task responsible for a certain activity and determining the corresponding amount of currentcy is not straightforward because of the two-dimensional distribution of energy. The framework addresses this problem by placing a device-specific payback policy for each of the managed devices. The payback policy determines when and how much currentcy is to be deducted to pay for use of the device. For example, the disk payback policy may share the spin-up cost by all subsequent accesses, and it may try to spread out the payment during the timeout period prior to the spin-down of disk.

If a task runs out of currentcy, no more energy consumption should be allowed. A task without currentcy is put into a wait queue and not eligible for further scheduling. This is also the procedure by taken by many operation systems when a task is blocked by a conventional resource. The task blocked by currentcy will be re-activated when it has currentcy again, usually given by the allocation module at the start of the next epoch. By using one common model for energy consumption on all devices, energy tradeoffs among

devices become explicit; currentcy spent on disk activities is no longer available for CPU cycles.

2.3.5 Currentcy Model Beyond the Framework

Many of the previous works of reducing energy consumption are on a per-task or per-device level ([10, 11, 18, 45, 53]). These works are orthogonal the work of this thesis and can be easily integrated into the energy management framework [2]. Furthermore, the framework can assist such local energy saving efforts by its global view of the system. In particular, currentcy is the common language used by different components of the framework. Currentcy is a powerful tool for sharing energy information and it is not limited to the use within the framework.

Currentcy can be used in many ways to communicate between the framework and other objects outside of the framework. Only a few of them are listed here. First, currentcy information is an indication of the energy resource level, which can make energy-performance tradeoff explicit for local decision makers. For example, if the system has enough energy to achieve the target lifetime, as indicated by the currentcy level, there is no need for local managers to trade performance for energy. Secondly, as a requirement to access devices, the currentcy can be used to predict a task's future activities. For example if there is no currentcy left in a task, devices can be safely put into power-saving mode till the next epoch. Thirdly, with detailed accounting information, a task can review its energy consumption on different devices and change its performance mode accordingly to best adapt to the environment. Finally, currentcy can be manipulated to formulate policies for certain purposes. For example, a word processor can set aside certain amount of currentcy for the running of a low-priority background spell checker.

2.4 Design and Policy Spaces

This chapter has proposed the currentcy model and the energy management framework. To manage the energy resource, policies for the management modules are still to be explored. This section briefly outlines the design and policy spaces. Detailed discussion is presented in later chapters.

1. Energy Accounting. Accurate accounting is the prerequisite for energy management. The currentcy model has been introduced, but the exact mechanism of energy accounting with the currentcy model is still untouched.

Placement of energy accounting. Different devices have diverse energy consumption characteristics and need to be handled differently. Thus energy accounting should be done at the device level and be implemented independently for each device. The accounting for the energy resource may be able to leverage existing infrastructure of traditional performance oriented device management, which is also implemented on a per-device basis.

Data acquisition. The next problem of energy accounting is how to acquire the data of energy consumption of each device. Questions include what are the existing supports for acquiring energy consumption information, how to harness these supports, and what are the desired supports from hardwares.

Payback policy. One goal of the Payback policy is to debit energy "fairly" to tasks. However it is sometimes difficult to define the "fairness". Another issue with the payback policy is its impact on energy efficiency. One such example is that if a large amount of currentcy is debited at the beginning of the disk spin-up, the task may be forced to go idle as the result and waste the energy of keeping the disk spinning. One possible strategy is to charge

currentcy at the end of disk access session after the disk is spun-down.

2. Achieving Target Lifetime. In the energy management framework, this goal is achieved by limiting currency allocation of each epoch.

Per-epoch allocation level. The per-epoch currentcy availability is based on the goal of target battery lifetime. Commonly used models of battery lifetime assume a constant power consumption, thus a power consumption limit is imposed that translates directly into the currentcy allotment every epoch.

Epoch length. Dividing time into epochs serves to throttle the energy consumption rate and has an impact on application behavior. Epoch length determines the granularity of currentcy allocation. Long epoch time provides larger allocations and the ability to spend currentcy in a more bursty fashion. Shorter epoch time may smooth the consumption rate at the risk of increased allocation overhead.

Dealing with errors. Errors exist inevitably in any real system. In the energy management system, errors can also be introduced by many sources such as the battery model or the device models etc. These errors may deviate the system from goals such as achieving the target lifetime. There are many possible approaches to make the system be robust to errors, from the online correction of every error source to the adjustment of only the overall energy consumption.

3. Resource Sharing. To provide differentiated service, the first thing is to define a mechanism of resource sharing. Resource allocation and resource consumption scheduling should all be consistent with the mechanism. There are many kinds of such mechanisms in different systems. For example, proportional sharing is widely used in general purpose systems, and admission control based mechanism is commonly used in real-time systems.

Handling of surplus currentcy. When a task's accumulated currentcy exceeds the cap, what happens to the surplus currentcy? Choices include forfeiting the surplus amount of currentcy, adjusting the cap, or distributing it to other tasks.

Energy consumption scheduling. One question is how to utilize the currentcy model to coordinate the energy consumption of devices to be consistent with the resource sharing mechanism. Another question is that for each device, what is the relationship between the energy oriented scheduling and the traditional performance oriented scheduling, and how to resolve if they conflict.

4. Interaction between Framework and Devices. The previous discussion on device management focuses on the requirement imposed by the energy management framework. It is also interesting to investigate how the framework could provide opportunities to improve devices' energy efficiency with currentcy-aware management. A few interesting ideas are listed here:

Subaccounts. Earmarking portions of a task's allowance for use with a particular device or by a particular thread within the resource container may require richer API support.

Bidding. The task may offer a price it is willing to support for access to an energy consuming resource. The bid does not necessarily imply that the task will be debited that amount for an activity.

Debiting. Instead of halting a task without currentcy, the system may allow the task continue to run in deficit for some amount of time in an effort to improve energy efficiency. One associated question is how should the task pay it back?

Pricing. The energy price of an activity is a way to encode thresholds in terms of currentcy and may interact with bids (e.g., in a negotiation protocol). Pricing may be decoupled from debiting to enforce threshold levels without skewing accurate accounting for the resource. Pricing may also encode the power state of a device (e.g., the price of a disk access is discounted when the disk is already spinning and no spinup is required).

5. Interaction between Framework and Applications. One extension of the framework is to take into consideration the performance impact of energy management. This is another area with lots of interesting topics.

Acquiring performance information. Applications of different types have different metrics for their performances. One application can even have multiple metrics. The challenge is that performance metrics are application specific and usually difficult to be implied by the OS.

Energy based admission control. To put the limited energy resource to good use, energy based admission control can prevent a task that can not meet its minimal energy requirement from getting into the system.

Resource Negotiation. The key idea here is that an application's performance and the energy consumption rate sometimes don't have a linear relationship. A task may negotiate with other tasks or with the system for an energy allocation different from the allocation imposed by the energy management framework to achieve a better performance/energy ratio.

2.5 Summary

Energy is a global resource. Its management is also a global effort that involves the management along three dimensions, the time dimension, the application dimension and hardware dimension. This chapter proposes the currentcy abstraction as the unified model for explicit energy management across the system. The currentcy-based energy management framework is also introduced in this chapter that manages the energy resource through the management of currentcy. At the end of this chapter, the management components of the framework are described and the management policy spaces are outlined. A prototype system that implements the energy management framework will be introduced in the next chapter.

Chapter 3

ECOSystem Prototype and Experimental Methodology

To evaluate the currentcy based energy management framework, and to evaluate energy management policies, a prototype system, called ECOSystem (Energy Centric Operating System), is implemented on an IBM Thinkpad laptop. It is based on the Linux operating system assuming a battery powered mobile environment running general purpose applications. This chapter introduces the ECOSystem prototype and the experimental methodology, including ECOSystem data structures, ECOSystem interaction interface, the hardware platform, and the application set for the experiments.

3.1 ECOSystem Implementation

ECOSystem is based on Linux kernel version 2.4.0-18. This section describes the implementation details of the management framework. The specific policies, policy implementations, and evaluations will be discussed in later chapters.

3.1.1 Resource Container

ECOSystem re-implements a subset of the resource container abstraction [19, 4] as the mechanism of currentcy management. The resource container abstraction separates the resource management entity from the thread of execution. For example, when a hard disk driver accesses the disk on behalf of a process, the process has to pass its resource container to the driver to pay for the cost of the access.

In ECOSystem, several processes may share one resource container. The implementa-

Container Operation	Description
container_new	Generate one container
container_attach	Attach a process to the container
container_detach	Detach a process from the container
container_SetParameters	Set parameters such as resource priority etc.
container_merge	Merge two containers, including the resource priority
container_separate	Seperate two containers including the resource priority
container_StartAccounting	Start gathering statistics of the container
container_StopAccounting	Stop accounting and printout the statistics

Table 3.1: Resource Container Operations

tion of resource container in the ECOSystem adds a new link for every process that points to the associated resource container. Resource container is the entity of management, containing information such as the priority, available currentcy, currentcy usage statistics and links to all the associated processes etc. The supported resource container operations are listed in table 3.1. The resource container abstraction may support other features such as multiple containers per process or priority inheritance, which are not implemented currently.

When the first process of an application is started by the user, a resource container is generated for the process automatically by calling $container\_new$ and then $container\_SetParameters$. Any further process spawned by the application will be simply associated to the same container by default using the function $container\_attach$. However, a new container can be created from the shared container by calling $Container\_Separate$.

Once a process is terminated, the container will be updated with the function $container\_detach$ and the number of associated processes is decreased by 1. The container will be destroyed if all of its associated processes are terminated. From now on, the term $task$ is used to refer a resource container and all the associated processes. The two functions $container\_StartAccounting$ and $container\_StopAccounting$ listed in the table are used to gather statistics about the container during an experiment.

3.1.2 Currentcy Management Data Structures

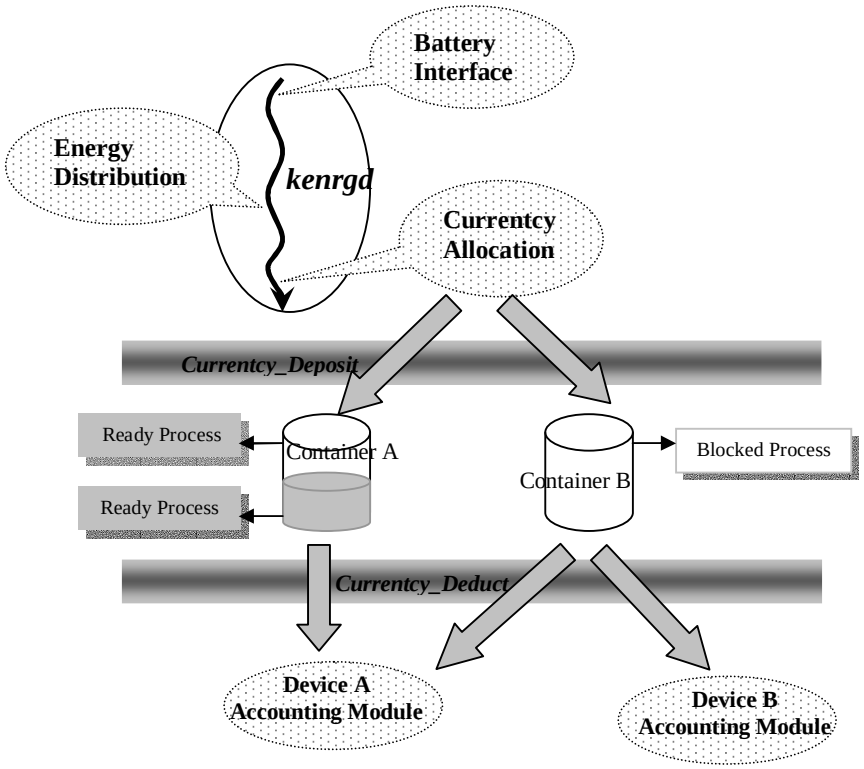


Figure 3.1: ECOSystem Data Structures for Currency Management

This subsection introduces the data structures that implement the currentcy management framework, as illustrated in figure 3.1. Function modules are also illustrated in the figure to show how they are incorporated in the framework data structure, but their policies are not discussed here. ECOSystem has a new kernel thread, $kenrgd$, that wakes up every epoch. During its execution, it will call three modules: the battery interface, the energy distribution module and the currentcy allocation module. Of the three modules, the battery interface is first called for acquiring battery information and then the energy distribution module is used to decide how energy is consumed over the battery lifetime. Finally, currentcy is generated for the epoch and given to containers by the allocation module.

There are two function calls and a wait queue added in ECOSystem to manage cur-

rentcy. One of the functions is $Currentcy\_deduct$ called by device accounting modules. It subtracts a certain amount of currentcy from the container and checks the balance. In figure 3.1, container B is out of currentcy. All associated processes are therefore marked ineligible for scheduling, taken out of the ready queue, and put into the wait queue. The other function $Currentcy\_deposit$ does the opposite and is called by the allocation module. If a container is out of deficit with the allotment, all associated processes will be awakened and made eligible to be scheduled again. In the figure, container A has currency in it and all associated process are marked $ready$.

3.1.3 Kernel Input/Output

In ECOSystem, all policy modules are implemented as separate kernel modules that can be installed and uninstalled independently without rebooting the machine. One difficulty with kernel programming is that the user can not interact with the kernel directly, the kernel can only be accessed through system calls. Adding and changing system calls frequently are very cumbersome for the experiments. ECOSystem supports an input interface that intercepts the name and the parameters of the binary to be executed. For instance, when the user types in "$ECOS\_add\_currentcy\ 6\ 100$", the OS will intercept and pass the command line to a kernel module that matches it against predefined character strings. In this case the user input will be mapped to a kernel function that adds 100 currentcy to a container of ID 6. Other predefined commands include setting target battery lifetime, container manipulation commands, policy module control commands and debugging operations such as step-by-step execution.

All information generated by the kernel is dumped through the Linux proc file system. Kernel messages are put into a circular memory buffer in the kernel space that maps to the proc file /proc/kmsg. The proc file is visible to the user just as a regular file. By Linux default, messages in /proc/kmsg are retrieved and cleared by a thread $klogd$, which wakes

up periodically and logs the messages to the disk. In ECOSystem the thread $klogd$ is disabled. Messages are retrieved only after an experiment to avoid activating the disk. The circular buffer size is increased from the default size of 16kB to 4MB to be big enough for all messages generated during an experiment. The laptop has a total of 128MB memory.

3.2 Hardware Platform Power Characteristics

While the management framework and ECOSystem prototype are general for a wide range of hardware platforms, ECOsystem is implemented on an IBM laptop, model Think Pad T20. Choosing a laptop platform has many advantages, such as stable software environment, fast kernel programming and debugging, available data from manufacturers, more applications to choose from and easy power measurement. However it also has some disadvantages. For instance, compared to PDA type devices, a laptop's idle power consumption is very high, which means less flexibility in controlling battery discharge rate. Also the CPU in the laptop has very high power consumption, overshadowing the power consumptions of other devices. The power characteristics of the platform do affect some of the experiment results, and are described in this section. The applications chosen for the laptop platform are introduced in the next section.

3.2.1 Overview

ECOSystem prototype currently has 3 managed devices: the CPU, hard disk, and network interface. It is possible to add more managed devices into ECOSystem, but these should be adequate to demonstrate the ability to unify multiple components under the currentcy model.

The system's base power consumption is measured to be 13.3W, including the base power states of all managed devices and the power consumption of all other devices. For instance the LCD display is always set at the highest brightness level. It has a power

consumption of about 4W and is a part of the base power consumption. To measure power consumption a Fluke multi-meter and micro-benchmarks are used to exercise each device individually. For some devices, the power consumption data from data sheet specifications is also used.

3.2.2 CPU Power Consumption

The CPU of the laptop is a 650MHz Pentium-III processor with the $Speedstep$ feature disabled to simplify the management space. When the processor is idle, the OS can issue a HLT instruction to put it into a low power $AutoHALT$ state, causing the clock to be stopped to most of the CPU's internal units [29]. While in the $AutoHALT$ state, the CPU can still service all requests by returning to the active state automatically. Other power states available to the pentium III processor(stop-grant, sleep and deep sleep state) require chipset support and may not be available to every system. Moreover these power states can not respond to interrupt automatically and incur relatively long transition delay to the active state, thus not suitable for the fine granularity control of CPU power considered in ECOSystem.

The $AutoHALT$ state is the base CPU power state of ECOSystem. When active, the CPU has an additional power consumption of around 12 watt to 20 watt, depending on the workload. For instance, it is measured that the CPU consumes about 15 watt when executing a loop of integer add operations.

3.2.3 Hard Disk Power Consumption

The ATA standard, also known as IDE interface, is a popular interface for today's hard disks. It defines three power saving modes (idle, standby and sleep) for the hard disk in its power saving feature set [52]. The disk is still spinning in the idle mode, but is spun-down in the standby mode. In the sleep mode, the disk can not accept any command except the

Travelstar 12GN	Typical Power	Corresponding ATA Mode
Performance Idle	1.85W	Idle
Active Idle	0.85W	
Low Power Idle	0.65W	
Standby	0.25W	Standby
Sleep	0.1W	Sleep

Table 3.2: Power Saving Modes of IBM Travelstar 12GN

$restart$ signal. The power saving feature set also includes the command for the OS to inquire the power mode of a disk and the commands to put it directly into any of these modes. In addition, it supports a built-in timeout based management scheme to spin down an idle disk with the timer set by the OS.

Unfortunately, the hard disk in our laptop, IBM Travelstar 12GN, has more power states than the ATA standard [28]. As shown in the table 3.2, the $idle$ state defined in the ATA is divided into three states for the IBM hard disk. Furthurmore, the transitions between these states are managed by an unknown internal algorithm that cannot be manipulated through the ATA interface. This complicates hard disk energy accounting since it prevents the OS from knowing the true power state of the disk. The standby mode is the deepest power saving mode set by ECOSystem and thus the base power mode of the hard disk, because it consumes significantly less power and can still accept access commands.

3.2.4 Wireless Network Interface

The network interface used in our system is an Orinoco Silver wireless PC card that supports the IEEE 802.11b standard. This card can be in one of three power modes: Doze (0.045W), Receive (0.925W), and Transmit (1.425W). IEEE 802.11b supports two power-utilization modes: Continuous Aware Mode and Power Save Polling Mode. In the former, the receiver is always on and drawing power, whereas in the latter, the wireless card can be in the doze mode with the access point queuing any data for it. The wireless card will wake

up periodically and get data from the base station. In the Power Save Polling Mode, the wireless card consumes a small fraction of the energy compared to the Continuous Aware Mode and most of the power is consumed by sending or receiving data for the user application. In the ECOSystem prototype, we always use the Power Save Polling Mode with the maximum sleep time set to 100 milliseconds as the base power state.

According to 802.11b, data retransmission may occur at the MAC layer as the result of data corruption. Data retransmission can consume additional energy invisible to the OS and can affect the accuracy of our energy accounting. In our tests, we enable the optional Request-to-Send/ Clear-to-Send (RTS/CTS) protocol at the MAC layer for transmissions larger than 1024 bytes to reduce the chance of collision. The MTU (Maximum Transfer Unit) is 1500 bytes in our system.

3.3 Experimental Methodology

This section introduces the methodology to evaluate the framework, including the metrics to evaluate the goals, the method of measurement, and the application set used in the experiments.

3.3.1 Metrics and Performing Measurements

In chapter 1, a set of goals is proposed to be achieved with ECOSystem. Table 3.3 lists the metrics used to evaluate each of these goals and the metrics to evaluate the system's impact on application's performance. These metrics can be roughly classified into three categories.

The first category is related to the system-level energy consumption regulation, which is a major concern of the energy management framework. The approach to measure the energy consumption of the system, each task and each device is introduced in the next

Category	Metric	Goal
Global Management	Lifetime achieved (second)	Achieving target lifetime
	Remaining battery capacity (mJ)	Reducing residual energy
	Energy consumption of each task (mJ)	Differentiated energy consumption
Local Management	Energy per device access (mJ)	Energy effi cient scheduling
	Scheduling delay variation (standard deviation etc)	Smooth scheduling
Category	**Metric**	**Purpose of Evaluation**
Application Impact	Amount of work done (application specifi c)	Application effi ciency
	performance compared to non-throttling (application specifi c)	Performance degradation by throttling

Table 3.3: Metrics in ECOSystem Experiments

chapter on energy accounting. The battery capacity information is retrieved by inquiry through standard battery interface, described in the next chapter.

The second category of local management includes the part of OS efforts for better device management or better service to applications within the framework and without violating the global goals. Energy efficiency scheduling tries to leverage the power characteristics of the hardware device to improve the access energy efficiency. Smooth scheduling tries to degrade the application's performance gracefully under the energy constraint.

The third category includes the metrics to evaluate the application's performance affected by the energy management. The amount of work done during the battery lifetime is used to measure the application's ability to accomplish its intended goal. The performance degradation is used to directly measure how the energy constraint affects the user-perceived performance. However the performance metric is usually application specific. Thus a set of applications typical for a mobile environment is selected and for each of these applications, an evaluation metric is defined that is believed to correlate with user-perceived performance. As shown in table 3.4, the real applications used are Netscape, realplayer, x11amp, gqview and ijpeg from the SPEC95 suite. Detailed explanation of the applica-

Application	Metric	Description
Netscape	Web page display delay	Web browser
gqview	Image display time	Image viewer
x11amp	Playback time	MP3 player
RealPlayer	Playback time	Video player
ijpeg	Time to process input fi le	SPEC2000 benchmark, image encoding

Table 3.4: Applications

tions and the metrics is presented in the remainder of this section. In addition to these real applications, microbenchmarks is also used for targeted evaluation of various system components.

3.3.2 Application Set

This subsection introduces the applications used for evaluation. For each application, a brief introduction of the application is presented, followed by the performance metric, measurement method, execution environment and power consumption characteristics.

Netscape

The first application, netscape, is representative of an entire class of interactive applications where a user is accessing information. The performance metric for netscape is the time required to complete the display of a web page. It is assumed that the page must be read from the network and that the netscape file cache is updated, so all three of the managed devices are included (CPU rendering, disk activity, and a network exchange).

The performance data is obtained by inserting several lines of java script code into the visiting web page. A few lines of code is also added in the web page that cause the Netscape to reload the page after 5 seconds. The time between page requests is to model the user's think time. The think time has the effect of allowing some amount of currentcy to accumulate between events.

Gqview

The next application, gqview is an image viewer that can be set to an auto-browse mode, in which all files in a directory will be automatically displayed sequentially. In the experiments all images are identical 0.5MB jpeg files. A 10 second timer is set between the display of each file to model the user think time. All disk buffers are flushed before each test.

The performance metric for gqview is the time to display an image. The time is determined by the input image size, format, and the rendering algorithm. Gqview is an open source program. Several lines of code is added in its source code to print out the time used to display an image.

X11amp

X11amp is an open source audio player. This is representative of a popular battery-powered application with user-perceived quality constraints. The metric for this application's performance is the playback time of a song. Any slowdown in playback manifests itself as disruption (e.g., silence) in the song, which is very annoying for most users. Since each song has a specific playback time, the slowdown of the x11amp is measured by comparing the actual time to complete the song against the normal playback time.

During an experiment, x11amp is configured to play an MP3 song from the hard disk. Its execution consists of both CPU activities and disk accesses. Before each test, all disk buffers are flushed, forcing all data to be read from the disk. The core of x11amp is the decoder. When playing an mp3 file, it is woken up when the sound buffer is about to be empty. It then decodes the mp3 input steam to fill up the sound buffer and then goes back to sleep. X11amp does not need many CPU cycles to decode the MP3 stream with a pentium III processor. This is reflected in its scheduling that it always gives up the scheduling

quantum voluntarily.

Realplayer

Realplayer is a popular video player. The performance metric for this application is also the actual playback time, measured against the playback time of the input video clip. During an experiment, the realplayer plays a stream of video over the wireless network and has high demands for all three managed devices.

Its CPU demand is determined by the encoding method and the encoding fidelity of the input stream, and is adjustable by choosing the window size and the playback quality. The bandwidth demand is determined by the bit rate of the stream. When there is not enough bandwidth, realplayer has to wait for the input buffer to be filled up, resulting in long pauses and increased playback time. If the available bandwidth exceeds the needed bandwidth, its behavior seems to depend on the network streaming protocol. In the case of direct HTTP connection to the media file, it was found that the realplayer always takes up all the available bandwidth and caches the bytes received to the hard disk.

Ijpeg

The final application, ijpeg, is computationally intensive and representative of digital image processing. It is run in a loop to continuously execute the SPEC command line, compressing an image from the reference data set (SPEC command line options: -GO.findoptcomp vigo.ppm). Unless otherwise specified, ijpeg is configured to process the same image in each run by default and the image will be cached in memory without activating the hard disk. Ijpeg can also be configured to use the hard disk by processing different images. To ensure regularity, identical images (vigo.ppm) are used differing only in the file name. The performance metric for ijpeg is the execution time to compress one input file, or simply the CPU utilization because they have a linear relationship.

3.4 Summary

A prototype system, ECOSystem, is implemented to evaluate the currentcy-based energy management framework. ECOSystem is based on Linux kernel. The data structures, including the resource container implementation for currentcy management, the new kernel thread for invoking the management components, and the interface for kernel input/output are introduced in this chapter. With the management components unified by the currentcy model, managing the energy resource is not drastically different from the management of other resources. For instance the resource container abstraction can also be used to the management of other resources, and the approach of blocking the out-of-quota task is the common approach in an OS.

ECOSystem is implemented on a laptop platform. The power characteristic of the platform and the three managed devices (CPU, hard disk and 802.11b card) are described.The applications running on ECOSystem are also introduced in this chapter. These applications are typical to a mobile system platform and each application has its specific metric for its performance. The configuration and the execution environment for each application are also described.

Chapter 4

Energy Accounting

Energy accounting in ECOSystem involves both hardware support and the OS efforts. At the hardware level, the energy consumption of each individual device needs to be measured. At the OS level, a mechanism is needed to break down the hardware energy consumption and attribute them correctly to application containers. This chapter investigates available support at the hardware level and introduces the embedded energy model approach for energy accounting.

4.1 Energy Accounting with Built-in Gauge

Support for online energy accounting is very limited in existing hardware. Previously, researchers have used either external assistances such as a multi-meter synchronized to target system activities [22, 21, 48], or intrusive approaches such as separating power planes specifically for the purpose of measurement [5]. ECOSystem, however, looks for support that can be incorporated in a working system. In the hardware platform of ECOSystem, the battery has the built-in capability of energy measurement. This section discusses the possibility of utilizing this built-in gauge for the purpose of energy accounting.

4.1.1 Battery Management and Battery Support for Energy Accounting

Today's battery not only has the self-measurement capability, it also has standard communication interface for OS inquiry. Important information related to the management of battery device includes the remaining capacity and remaining lifetime. However, this section does not focus on the management of battery itself, instead it investigates the possibility of harnessing the battery as a built-in multimeter. This is possible because when the

system is powered only by battery, the system's energy consumption during a time interval equals to the battery's capacity drop during the interval, and the system's instantaneous power consumption equals to the instantaneous discharge rate of the battery. The rest of this section introduces the battery interfaces and the battery information available from these interfaces, followed by a discussion on utilizing the energy measurement capability for the purpose of energy accounting.

4.1.2 Introduction to Battery Interfaces

There are three popular communication interfaces to access battery information: the Smart Battery interface [6] and the Control Method battery interface [1] defined in the ACPI standard, and the APM interface for battery information [3]. Before going into each battery interface, a brief introduction to APM and ACPI is provided.

APM and ACPI

Both AMP and ACPI are intended for system wide energy management. APM(Advance Power Management BIOS Interface) is a widely used standard that manages the power state of devices in the APM BIOS. APM BIOS can interact with the OS and applications, allowing them to be involved in the energy management [3].

ACPI is a newer standard. It is established for OS-directed configuration and power management that integrates changing energy management technologies with a standard interface [1]. This is done by embedding AML(ACPI Machine Language) into ACPI compatible devices. Basically, AML is a small program inside the hardware implementing the function calls defined in the ACPI. The OS can simply invoke the corresponding AML code for a certain function to avoid the low-level details of the hardware.

APM Battery Interface

The APM standard includes an interface for inquiring the battery information, which is described in table 4.1 .

Data Field	Description
Battery Status	High/Low/Critical
Battery Flag	Charging/Discharging
Remaining Percentage of Full Charge	1-100
Remaining battery lifetime	Lifetime in seconds or minutes

Table 4.1: APM Interface Battery Information

As can be seen in the table, the APM battery interface contains the information of remaining capacity and an estimation of the remaining lifetime, but no information about the instantaneous discharge rate. The remaining capacity information comes in integer between 1-100.

Control Method Battery

An ACPI compatible battery must be either a Smart Battery or a Control Method Battery. CM Battery (Control Method Battery) allows the OEM to choose any type of battery and any kind of communication interface supported by ACPI. A CM Battery needs to declare itself to the OS and is accessed through AML.

Data Field	Description
Battery State	Charging / Discharging/Critical State
Battery Present Rate	Energy Draining Rate in mA/mW
Battery Remaining Capacity	In mAh/mWh
Battery Present Voltage	In mV

Table 4.2: Control Method Battery Interface for Battery Information

In the table above, the data field of $Present\ Rate$ returns a snapshot of the battery's discharge rate. It can choose one of two units for the returned data, either in mA or mW.

For the ECOSystem battery with rated voltage of 10.8V, mW is a much finer unit than mA (1mA*10.8V = 10.8mW). Similarly mWh is finer than mAh for battery remaining capacity information.

The battery in ECOSystem platform is a CM Battery. But the ACPI implementation as of Linux kernel version 2.4.0-18 is still experimental, errors occur when executing the AML code for the battery interface. To circumvent this problem, ECOSystem takes the approach of accessing I/O ports directly by analyzing the AML code. This approach also saves the overhead of AML execution.

4.1.3 Smart Battery

With Smart Battery interface, multiple batteries can be controlled directly by the OS through the SMBus (System Management Bus). Smart Battery Interface supports a large function set. Some of these functions that are relevant to energy accounting are listed in table 4.3.

Function	**Data Range**	**Description**
AtRateToEmpty	0 to 25534 min	Battery lifetime at a certain discharge rate
Voltage	0 to 25535 mV	Current Voltage
Current	-32768mA to 32767 mA	Charge or discharge rate
AverageCurrent	-32768mA to 32767 mA	One minuet rolling average of current
RelativeStateOfCharge	0 to 100 percent	Percent of remaining capacity
RemainingCapacity	0 to 65535 mAh or 10mWh	Remaining capacity at C/5 or P/5 discharge rate

Table 4.3: Smart Battery Interface for Battery Information

The amount of energy that can be extracted from a battery is affected by the rate of discharge. This is the reason why the function $AtRateToEmpty$ and $RemainingCapacity$ have to assume a certain discharge rate in their calculations, which is an improvement to the lifetime estimation of APM and CM battery interface. Another unique feature of the

Smart Battery interface is that all functions return data with the data granularity and accuracy specified. For example in the function $Voltage$, it is specified that the data granularity should be 0.2% of the design voltage or better, and the accuracy should be within 1.0% of the design voltage.

4.1.4 Accounting Requirements and Battery Interface

As shown in the tables 4.1, 4.2 and 4.3, today's battery interfaces can provide pretty complete information about the status of the battery. But certain requirements have to be considered when using it as the built-in "gauge" for the accounting purpose. In ECOSystem, energy accounting should be accurate, low overhead, and have the ability to differentiate accesses from different tasks. This subsection discusses each of these requirements and see if they can be met by the battery interfaces.

Accuracy

Both APM and CM Battery interfaces do not specify an error range with their returned results, thus the OS has no idea of their accuracy. For example, when examining the remaining capacity information returned by the ECOSystem CM Battery, even though it uses the unit of mAh, it is found out that its last digit is always zero, which means the lower bound of its error range is 10mAh.

Although it is not possible to measure the accuracy of every battery, data accuracy is upper bounded by the data granularity defined in the interface specification. For instance in the APM standard, the remaining battery capacity is returned in percent between 1 and 100. For a battery lifetime of 3 hours, its capacity reading will only change (decreased by 1 percent) every 1.8 minutes. This is too coarse-grained for the purpose of energy accounting compared to the 200ms CPU scheduling quantum of ECOSystem.

Remaining battery capacity returned by CM battery can be in either mAh or mWh.

But even using mWh, the finer unit of the two, the reading of battery capacity will only change every 0.36 second with a 10W power consumption. ECOSystem CM Battery uses mAh. Even if it were accurate up to 1mAh, the reading would only change every 3.89 second under 10W power consumption. The function $RemainningCapacity$ from the Smart Battery interface has the unit of 10mWh, even more coarse-grained than the CM battery interface.

Overhead

Energy accounting in ECOSystem needs data in real-time and with low overhead, which is a challenge in using real measurement for fine-grained control. Accessing an energy measuring component is likely to be slow. In the Smart Battery system, the OS communicates with batteries through a two-wire System Management Bus. According to the specification, the Smart Battery controller alone can delay any data request for up to 25 ms in addition to all other overhead. A 25ms delay equals a 12.5% overhead every 200ms CPU quantum.

There are different ways of accessing CM batteries. For the ECOSystem CM battery, it is connected to an embedded controller which is then attached to the PCI bus. With the direct I/O accessing approach, the overhead of AML execution is avoided when acquiring the package of battery information described in table 4.2. The accessing time is measured to be around 2 ms. However this overhead does not include the actual measuring cost. It is found out that the battery will sample the discharge rate automatically at a fixed interval of 20ms and return the last sampling result for any inquiry during the interval. Thus the data returned may not be the “true” instantaneous discharge rate at the moment of inquiry.

Accounting for Process Activities

Previously, the limitations of existing battery interfaces with respect to data accuracy and accessing overhead have been discussed. However even if better battery-related data were available, it can only be used to measure system's total energy consumption. Other accounting issues would remain related to accurately attributing power/energy usage because of the two-dimensional energy distribution.

In an experiment to test the battery's ability to differentiate activities from two processes,an environment is set up with only synchronous activities. There are two CPU-only synthetic benchmarks that individually produce a distinct, stable power consumption profile. When scheduled together, the reported power values can be any number between the two stable power consumptions. In conclusion, the battery can not differentiate the two processes even when no asynchronous activity is involved.

4.2 Embedded Energy Model Approach

The previous section has discussed the energy accounting requirements and the limitations of existing hardware support. Based on the discussion, the embedded model approach is proposed for ECOSystem energy accounting. The embedded model approach has three components: device energy model, device specific payback policy and task tracking infrastructure. The device energy model is used to estimate the energy consumption, the task tracking infrastructure is used to identify activities from processes, and the payback policy is used to attribute the energy consumptions to containers.

With the embedded model approach, the energy accounting overhead decreases with the speed of the CPU. Energy model building may be helped by manufacturers, for instance some form of embedded support in the hardware. It is also possible to adjust or fine-tune the energy models by real energy measurement, for instance from the battery interface.

The measurement can take place in the background and no delay is incurred.

4.2.1 Energy Accounting for CPU

When building a device energy model, the level of detail of the model depends on the information that is available to the OS. The power status of the device must be visible to the OS, either by OS inquiry or event notification.

CPU Energy Model and Activity Tracking

ECOSystem uses a model that assumes that the CPU draws a fixed amount of power (currently 15.55 W) for computation. This was established by measuring the power while running a loop of integer operations. The current model is very simple (e.g., CPU halted or active) but the embedded energy model approach can support finer-grain information such as using event counters to track processor behavior as suggested by Bellosa [7]. ECOSystem could also benefit from component specific "gas gauges" that accurately measure energy consumption.

Tracking task responsible for CPU activity is relatively simple because all CPU activities are synchronous, meaning that the activities can be simply attributed to the current running task.

Sampling Based Accounting

In Linux, a sampling based approach is already in existence for performance oriented CPU accounting. The sampling is done at the granularity of timer interrupt (every 10ms in ECOSystem platform). At the beginning of scheduling, every process in Linux is initialized with a $count$ value according to its priority. Every timer interrupt, the current running task is sampled and the $count$ value is decreased by 1 for the CPU usage since last sampling. The task will be switched out when its $count$ reaches zero.

This sampling approach is simple and low in overhead. The problem with this approach is that the unit of accounting is 10ms and it will not correctly account for short-computation processes. For instance, the x11amp requires less than 10ms to compute when it is woken up (section 3.3.2). When it is running with another long-computation process, very often it will be scheduled right after a timer interrupt when the $count$ value of the other process is 0. And because it always gives up the CPU before another timer interrupt, it will not be charged, even if it had used 9ms out of the 10ms quantum. This is not a big problem for performance oriented accounting because short, interactive processes are usually intensionally favored. But it is not acceptable for energy accounting because the x11amp may be able to accumulatively consume a large amount of energy without being accounted.

4.2.2 Hybrid CPU Accounting

ECOSystem's payback policy for the CPU is a hybrid of sampling and process switching accounting. Accounting is activated when there is a switch of processes to avoid the "round-down" problem above. A function $gettimeofday$ is used to time-stamp the process instead of using the $count$ variable. This function utilizes CPU internal counter and returns time in nanoseconds. The accuracy of this function depends on the hardware platform but is enough for CPU energy accounting. The overhead of this function is measured to be less than $100ns$ in ECOSystem hardware platform.

In addition to process switching, accounting is also done at every timer interrupt. If a task is found to be out of currentcy during its quantum, it will be preempted early. This approach of setting up "checkpoints" for energy accounting prevents the long-computation processes from continuing to run with insufficient currentcy. Preempting process timely is also good for the execution of the process because being preempted earlier means less waiting each time it is paused and thus smoother execution on average.

	Cost	Time Out (Sec)
Access	1.65mJ/Block	
Idle 1	1600mW	0.5
Idle 2	650mW	2
Idle 3	400mW	27.5
Standby (disk down)	0mW	N/A
Spinup	6000mJ	
Spindown	6000mJ	

Table 4.4: Hard disk power state and time-out values

4.2.3 Hard Disk Energy Accounting

When accessing data, the hard disk consumes about 2.0W to 2.3W [28], which is relatively small compared to the CPU's 15W plus power consumption. But when in idle state, the hard disk may still drain energy continuously to keep the disk spinning, making it a big energy consumer even for applications of infrequent disk accesses.

Energy Model

The unknown internal management algorithm of our IBM TravelStar 12GN prevents ECOSystem from building an energy model that match its actual behavior. Therefore, the disk's power consumption is approximated using a timeout based model derived from typical hard disks. Table 4.4 shows the values used in the model. It is well known that the energy cost to spinup the disk is high. For the IBM TravelStar, it is also observed that the power consumption increases when it tries to spin down the disk. Thus the spindown cost is set to a fairly large value of 6000mJ for the disk model. The disk model is set to spin down after 30 seconds. To achieve comparable effects on timing, the Travelstar hard disk is also set to spin down after 30 seconds.

Activity Tracking

Energy accounting for hard disk activity is very complex. The interaction of multiple tasks using the disk in an asynchronous manner makes correctly tracking the responsible party difficult. To track disk energy consumption, file related system calls ar instrumented to pass the appropriate resource container to the buffer cache. The container ID is stored in the buffer cache entry. This enables accurate accounting for disk activity that occurs well after the task initiated the operation. For example, *write* operations can occur asynchronously, with the actual disk operation performed by the I/O daemon. When the buffer cache entry is actually written to disk, an amount of currentcy is deducted from the appropriate resource container. Energy accounting for *read* operations is performed similarly.

The present implementation does not handle all disk activity. In particular, inode and swap operations are not addressed. The swap file system has its own interface and does not follow the vnode to file system to file cache to block-device hierarchy. For the reported results, such activity constitutes only a small fraction of overall disk activity (as reflected by the overall accuracy of the achieved energy allocation).

Payback Policy

Further complexities of hard disk energy accounting are introduced by the relatively high cost of spinning up the disk and the large energy consumption incurred while the disk is spinning. ECOSystem has implemented a reasonable initial policy to address this complexity. However, further research is clearly necessary.

The cost can be broken into four categories: spinup, access, spinning, and spindown. The cost of an access is easily computed by $\frac{active-state-power-cost(W)}{disk-access-bandwidth(KB/s)} * buffersize(KB)$. The energy consumed to access one buffer on disk is 1.65mJ on the ECOSystem platform. Since a dirty buffer cache entry may not be flushed to disk for some time, multiple tasks

may write to the same entry. The current policy simply charges the cost of the disk access to the last writer of that buffer. While this may not be fair in the short-term, the long-term behavior should average out to be fair. The remaining disk activities present more difficult energy accounting challenges.

The cost of spinning up and down the disk is shared by all tasks using the disk during the session defined by the period between spinup and spindown. It is charged at the end of the session and is divided on the basis of the number of buffers accessed by each task. It is also possible that the disk can just keep spinning. In this case if the disk has been up for over 90 seconds, the cost is charged at this moment and shared by all tasks that have been active over the last 90 seconds. It is assumed in the disk model that spinup or spindown takes 2 seconds and that the average power is 3,000mW, leading to a total energy cost of 6,000mJ.

The cost for the duration of time that the disk remains spinning waiting for the timeout period to expire (30 second minimum) is shared by those tasks that have recently accessed the disk (in essence, those that can be seen as responsible for preventing an earlier spin-down). This is done by incrementally charging the tasks that have performed accesses within the last 30 second window in 10ms intervals (timer interrupt intervals). On each timer interrupt, if the disk is spinning, the energy consumed during this interval, as determined by the disk power state and length of the interval (10ms), is shared among those tasks active in the last 30 seconds.

4.2.4 Network Interface Energy Accounting

Wireless network interface is an essential part of a mobile system. It can consume a large amount of energy if it always stays awake listening for incoming traffic. In ECOSystem the 802.11b card is set to power-saving mode in which it is only awake every 0.1 second to check for incoming packets and then goes back to sleep.

Energy Model and Payback policy

The 802.11b card in ECOSystem platform has constant power consumptions in sending and receiving mode. When set in power-saving mode, its energy consumption increases linearly with the number of bits transmitted. To calculate the energy for transmitting a single bit, one simple method is to divide the power consumption by the transmission bit rate. However, IEEE 802.11b standard supports several transmission rates, from 2M bps to 11M bps, depending on the signal quality. Thus the energy per bit can vary by a factor of 5. ECOSystem is able to monitor the status of the wireless network interface and adjust the calculation accordingly.

Currently, the energy accounting only works for TCP/IP. Energy accounting for the network interface is implemented in ECOSystem by monitoring the packet size transmitted and received. The size of the packets is measured at the lowest possible level, before the they go into the hardware buffer for outgoing packets and after they are retrieved from the hardware buffer for incoming packets. The packet size includes all the headers, CRC fields and paddings. These values are then used to compute the overall energy consumption according to the following equations:

$E_{send} = (sent\_bits * transmit\_power)/bit\_rate$

$E_{recv} = (received\_bits * receive\_power)/bit\_rate.$

The current energy model is based on the information that can be seen by the OS. More understanding of the hardware may lead to a more detailed model, that includes energy cost of hand-shaking with the base station or the cost of additional MAC layer header.

Activity Tracking

Network interface activities are all asynchronous. For outgoing packets, the sender's container needs to be passed to the device driver. For incoming packets, because the desti-

nation process is still unknown when the packet is received and the energy is consumed, ECOsystem takes the opposite approach of passing the energy charge up the hierarchy with the packet until the destination's container can be identified.

In ECOSystem, the energy consumption is calculated at the device driver. The socket structure and the TCP/IP implementation is instrumented in ECOSystem to track the task responsible for a particular network access. When a socket is created for communication, the creator's container ID is stored in the socket. For each outgoing packet, the source socket and hence the associated source task is identified at the device driver. For an incoming packet, the energy consumption for receiving the packet is computed and initially stored with the packet when it is received. The destination socket of this packet will be available after it is processed by the IP layer. Currentcy is deducted from the destination task at this moment. If packets are reassembled in the IP layer, the energy cost of the reassembled packet is the sum of all fragmented packets. This energy accounting approach for TCP/IP connections can also be applied to other types of protocols such as UDP/IP and IPV6. In IPV6, the destination socket may be available before being processed by the IP layer which can ease the job of task tracking.

4.3 Energy Accounting Experiment

This section starts the experiments by comparing the currentcy based accounting to the program counter sampling accounting. Then the currentcy based accounting is used to study the energy consumption characteristics of the applications in the evaluation application set, including x11amp, gqview and netscape.

4.3.1 Currentcy Model based Accounting

This experiment is to validate the energy accounting advantages of the unified currentcy model. In this experiment three synthetic benchmarks are used to exercise the disk, net-

App	Currentcy Model				Program Counter Sampling				Error
	CPU(mJ)	HD(mJ)	Net(mJ)	Total(mJ)	CPU(mJ)	HD(mJ)	Net(mJ)	Total(mJ)	(mJ)
DiskW	430	339,319	0	339,749	430	16	24	470	339,279
NetRecv	256,571	0	553,838	810,409	256,571	9,235	20,206	286 ,012	524,397
Compute	8,236,729	0	0	8,236,729	8,236,729	326,404	531,789	9,094,922	858,193

Table 4.5: Unified Currentcy Model vs. Program Counter Sampling

work and CPU, respectively, in well-defined and recognizable ways. The disk benchmark (DiskW) does little computation and writes 4KB of data to the disk every four seconds. The kernel daemon will flush these dirty buffers every 5 seconds and thus keeps the disk continually spinning. DiskW is the only one of the three benchmarks designed to touch the disk. The network benchmark (NetRecv) also performs very little computation, but continuously receives data at the maximum bandwidth of the network. NetRecv is the only one of the benchmarks involved in wireless communication. The final benchmark is a CPU-only batch job (Compute) capable of running continuously. The three benchmarks are executed simultaneously for 548 seconds. By design, NetRecv should be considered responsible for consuming at least .925 mW in the NIC (the cost of receiving) and DiskW should be considered wholly responsible for approximately 620 mW in the disk (averaging power states idle1, idle2, and idle3 from Table 4.4 over the 5 second flushing interval) for the duration of the experiment. Thus, DiskW should consume approximately 338,760 mJ just for the disk and NetRecv should consume 506,900 mJ just for the NIC by "back of the envelope" calculations. Similarly, it can be estimated that Compute, if run in a stand-alone fashion, should consume no more than 8,521,400 mJ (at 15 .55 W).

The energy accounting data can be collected from ECOSystem's currentcy model. For comparison, ECOSystem emulates a program counter sampling technique by charging all disk and network activity to the resource container of the task occupying the CPU at each observation.

Table 4.5 shows the energy accounting results for both the unified currentcy model and

App	CPU (mW) Power	HD (mW) Power	NIC (mW) Power	Total (mW) Power	Metric	Performance
Ijpeg	15550	0	0	15550	Process delay	2.45 secs
X11amp	17.15	773.59	0	790.74	Playback time	201 secs
Gqview	5901.81	612.45	0	6514.26	Image display delay	6.3sec
Netscape	3001.22	683.92	133.40	3818.54	Page load delay	3.12 sec

Table 4.6: Application Power Consumption Characteristics

the program counter based technique. Note that in the program counter sampling approach only the Total column would be reported, however the table shows the breakdown by device as captured by the model to illustrate the source of any discrepancies. These results match our expectations – the program counter approach does not attribute the energy consumption to the appropriate tasks. The Total energy consumption values for DiskW and NetRecv is significantly lower than their consumption for the disk and network devices alone and Compute is assigned a higher Total consumption. In contrast, ECOSystem appears to more accurately charge each task for its specific device utilization. The ECOSystem results reflect the engineered-in behaviors of these synthetic benchmarks. Although for this experiment the CPU dominates power consumption, alternative platforms with a lower power processor or an application that doesn't fully occupy the CPU will decrease the CPU power component and increase the relative error of program counter sampling techniques.

4.3.2 Energy Accounting of Several Applications

With the currentcy based accounting, it is now possible to study quantitatively how energy is consumed by target applications on the hardware platform. Table 4.6 shows the breakdown of power consumption on each device for several of the applications when running unthrottled. Running unthrottled means running alone on ECOSystem platform with all resources devoted to it and without energy constraint. At the last two columns, the table

lists the performance metric for each of these application (see table 3.4). When testing $Netscape$, it is used to load a CNN homepage of a certain date. To experiment in a controlled and repeatable environment, all related pages and images are copied into a local server so that the network abnormality is reduced to the minimal. The running configurations for other applications have been introduced in section 3.3.2.

One thing can be observed from the table is that applications have totally different demands for power. $X11amp$ only requires as little as 790mW to run smoothly, although it can be a very important application to the user. The power consumptions of $Netscape$ and $gqview$ depend heavily on the data input, in this case they both require significantly more power than $x11amp$. $Ijpeg$ can use all the CPU cycles and it needs a large power supply of 15,550mW to run unthrottled. These data agrees with the hypothesis for the explicit energy management that it is possible to limit system energy consumption while still guaranteeing the execution of the task important to the user.

When looking at the power consumption at each device, it can found out that even though the CPU has the biggest power consumption in all devices when it is active, it may not be the biggest power consumer for an application. $X11amp$ only spends 17mW on the CPU because MP3 decoding only requires a few CPU cycles. On the other hand it spends 773mW on the disk because it keeps the disk in high power modes by reading small chunks of data from the disk every several seconds. $Netscape$ and $gqview$ read and write more data than $x11amp$ on average, but they consume less power on the hard disk because their think time allows the disk to go down in deeper power saving modes.

4.4 Summary

This chapter studies the problem of energy accounting. The first challenge with energy accounting is to measure the hardware energy consumption. Existing hardware lacks the support for energy consumption measuring of individual device. Battery interface can be

used to measure system total power consumption. However, through our experiments, it is found out that existing battery interfaces fail to meet ECOSystem's requirement for hight accuracy and low overhead. This chapter proposes the approach of building an energy model for each device that estimates the energy consumption of its activities. Because of application's asynchronous activities, another challenge of energy accounting is to attribute the hardware energy consumption to the tasks responsible. A payback policy is proposed for each device that charges the energy consumption to the tasks responsible.

The energy models for the three managed devices are introduced in this chapter. Currently ECOSystem only has very crude models for the three devices to demonstrate the approach of explicit energy management. However, they can be improved with more accurate modeling technique.

In the experiment section, the currentcy-based embedded energy model approach is compared to the program counter sampling approach, which simply charges the total system energy consumption to the current running task. The experiment results show the advantage of the currentcy-based approach in accounting for asynchronous activities.

Chapter 5

Achieving Target Battery Lifetime

In this chapter, ECOSystem is ready to be evaluated with the first goal: achieving target battery lifetime. However this chapter assumes a scenario of only one running application to simplify the currentcy allocation and scheduling policies and focus on the problem of battery management. The general scenario of multiple concurrent applications will be discussed in the next chapter.

The problem of how to discharge the battery to achieve the goal of target lifetime is discussed in the section 1. The non-linear properties of battery is also discussed in this section. Section 2 introduces a set of policies that can be plugged into the management framework and control the battery discharge rate to the appropriate level. Section 3 analyzes the ECOSystem stability with control theory and proposes a feedback online adjustment technique to achieve the target lifetime even in the presence of device modeling errors. Finally, experiments and results are presented in section 4.

5.1 Discharging Battery for Target Lifetime

The previous chapter has discussed the possibility of utilizing the built-in gauge in the battery to measure device energy consumption. This section focuses on the management of the battery itself. In ECOSystem, the target battery lifetime is achieved by controlling the energy discharge rate from the battery. However, determining the appropriate discharge rate for the target lifetime is not as simple as it might look like, due to very different battery details and its non-linear electro-chemical properties.

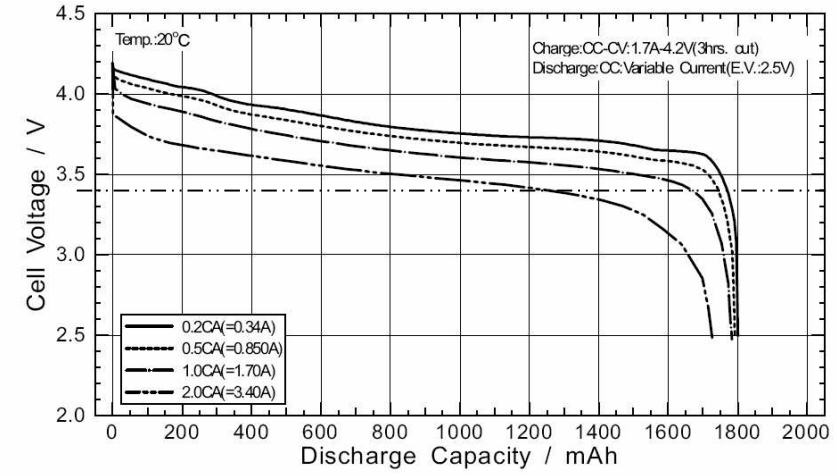


Figure 5.1: Rate Capacity Effect of Sanyo UF103450P Li-ion Battery

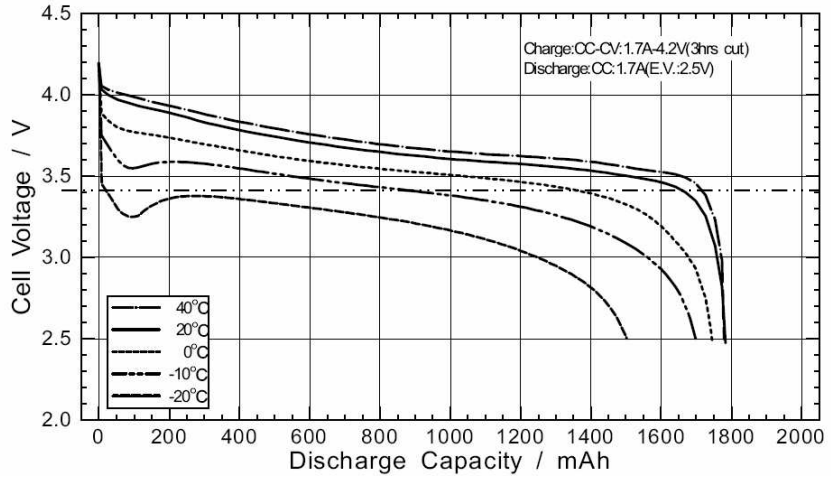


Figure 5.2: Temperature Capacity Effect of Sanyo UF103450P Li-ion Battery

5.1.1 Non-linear Battery Properties

Every battery is rated with two values, voltage and capacity. Ideally, a battery remains a constant voltage output, and then drops immediately to zero when it depletes. An ideal battery would also deliver a fixed amount of energy, which means that for a stable discharge rate of I, the battery lifetime T has a linear relationship with $1/I$ ($I * T = C$, where C is a constant). Estimating lifetime is easy with an ideal battery. For example, if an ideal battery can last 5 hours with 1 Amp discharge rate, then it should last 1 hour with the discharge rate of 5Amp.

Unfortunately, these ideal assumptions are not realistic, some non-linear battery properties have to be considered when managing the battery. Figure 5.1 shows the discharge rate characteristics of an example Sanyo Li-ion battery obtained from the manufacturer web site [46]. The Sanyo UF103450P Li-ion battery has a nominal voltage of 3.7V, but the actual voltage can be as high as 4.2V initially and it drifts down gradually. The voltage will decrease sharply as the battery is approaching empty. The system has to cut off the battery when the voltage is below a threshold value.

Also shown in the figure 5.1, the relationship between lifetime T and discharge rate I is non-linear. The faster the rate of discharge, the less total energy is delivered by the battery. This is called the *rate capacity* effect. The Sanyo battery has a nominal capacity of 1700mAh. If the cut off voltage is 3.4V, then the delivered capacity from the battery ranges from 1250mAh to 1750mAh with different discharge rates.

The $ratecapacity$ effect was described mathematically back in 1897 by Peukert with the equation

$$I^n * T = C \tag{5.1}$$

The exponent n in the equation describes the characteristic of the battery and is battery specific, usually it is between 1.0 and 2.0. Assuming $n = 1.2$, if a battery lasts 5 hours with

1Amp discharge rate, then with 5Amp discharge rate it can only last 0.725 hour instead of the 1 hour lifetime for an ideal battery.

The non-linear relationship between I and T complicates the management of the battery. Moreover, the delivered battery capacity is affected by environmental factors such as temperature. Figure 5.2 shows the discharge temperature characteristics of the Sanyo UF103450P battery. It can be seen that the temperature has a huge impact on the delivered capacity of battery. Assuming the cutoff voltage of 3.4V, the delivered capacity is decreased almost by half when the temperature decreases from 40^oC to -10^oC, and it can not function properly under -20 ^oC.

In addition to average discharge rate and temperature, the amount of energy delivered by a battery is affected by many other factors such as the battery type, the discharge waveform and the aging of the battery [15, 14, 38]. For instance pulsed power consumption can be helpful for the battery to "relax" and "recover" some of its energy. This is called the *recovery* effect. It is shown in [39] that different current waveforms of the same average value could result in 23% difference in the delivered energy from a battery.

5.1.2 Addressing Non-linear Properties

The modular architecture of the currency based energy management framework can support different approaches to address the non-linear battery properties. Two of these approaches are discussed below: the battery modeling approach and the ACPI approach.

Battery Modeling Approach

One possible approach is to build a model for these non-linear behaviors. The modeling based approach usually involves two phases: the calibration phase and the calculation phase. Because even identical batteries can vary as much as 20% in the capacity [8], the parameters should be calibrated for every battery. During the calculation phase, the work

load, including the temperature and discharge waveform, is input to the model. A more detailed model usually needs a more detailed description of the work load.

There are a wide range of battery models available. ECOSystem must consider the accuracy as well as the calibration cost and calculation overhead of these models. The Peukert's equation 5.2 can be used as a very simple model that only considers the *Rate Capacity* effect. The constant C and the exponential index n must first be calibrated. Given the estimated future average discharge rate, the battery lifetime can be estimated as

$$T = \frac{C}{I^n} \tag{5.2}$$

[35] models the Lithium-ion battery down to the electro-chemical level using Partial Differential Equations. This detailed model is proved to be pretty accurate, but it has lots of parameters to be calibrated and takes days to simulate. Faster models at the circuit level are proposed in [26, 23], but the simulations are still very slow. Also they are continuous models that require the discharge profile to be described in continuous functions.

A high level discrete-time battery model is proposed in [8] using VHDL. In [39], battery is modeled as a discrete time transient stochastic process. Their simulation time is still significant for online battery modeling, especially that with these discrete-time models, the battery discharge waveform must be measured and simulated in the granularity of microseconds. The data measuring and transmission could cost additional overhead.

Another difficulty with the modeling based approach is that some hardware details have to be parameterized into the battery model. Particularly, a DC/DC converter is widely used with the battery to stabilize the voltage and current output. It can consume 30% to 10% of battery energy because of its inefficiency [51]. Depending on how it is implemented, the DC/DC converter will also affect the discharge waveform of the battery.

ACPI Approach

A different approach taken by ACPI is that all these complexities should be dealt by the battery itself. The idea is that the battery itself is in the best position to know about all the details. First of all, the battery can directly monitor its status instead of relying on the accuracy of the model. For instance the Smart Battery is able to measure its own voltage, discharge rate and temperature. It can also keep track of some historical information such as averaged discharge rate and charge-discharge cycle count.

Secondly, some hardware details are easily accessible to the battery. It is also possible to build an embedded look up table from the manufacturer specifications such as the ones shown in the figure 5.1 and 5.2. For the CM battery in the ECOSystem platform, it is found out that it uses some prior knowledge about the battery to estimate the remaining capacity.

Finally, by processing locally and providing a standard interface, the battery relieves the OS from gathering battery information and hides battery technology details from the OS. The Smart Battery supports many queries such as prediction of fully charged capacity, prediction of remaining lifetime with current discharge rate and prediction of the time for the battery to be fully charged. In particular, the Smart Battery supports estimation of remaining battery lifetime at a discharged rate specified by the OS. This query is done in two steps. In the first step the OS submits a discharge rate i with the $AtRate()$ function. Then the OS can call the function $AtRateTimeToEmpty()$ for the estimated operating time with the discharge rate of i.

One problem with the ACPI approach is that the operation is limited by the interface. ECOSystem needs to estimate the appropriate discharge rate for the target lifetime, which is the reverse of lifetime estimation. With the Smart Battery interface, the OS can try different discharge rates with the function $AtRate()$ and $AtRateTimeToEmpty()$ to ap-

proximate the correct value until the estimated lifetime returned by the functions is close enough to the target lifetime.

Battery Management in ECOSystem

ECOSystem takes the ACPI approach because of its simplicity and low overhead. Another advantage with this approach is that the estimation accuracy can improve independently as more sophisticated self-monitoring or estimation techniques are built into battery. The disadvantage is that the system can not control the estimation accuracy, it has to accept whatever comes with the hardware. The system can not switch to a more accurate one as can be done with the battery modeling approach.

The battery in ECOSystem platform is a Control Method battery that only supports a very limited set of functions. Most importantly it lacks the discharge rate based lifetime estimation found in the Smart Battery interface. In the ECOSystem prototype, the discharge rate to achieve target lifetime can be simply calculated as

$$AverageDischargeRate = \frac{BatteryRemainingCapacity}{RemainingLifetime} \tag{5.3}$$

The discharge rate calculation in the equation 5.3 may introduce some errors because the remaining capacity reported by CM battery is based on an unknown discharge rate and the discharge rate is calculated with a simple linear function. This discharge rate calculation is limited by the battery type of the platform. More accurate calculations can be easily used in ECOSystem when they are supported by the hardware.

The error introduced by the calculation may affect the achieved battery lifetime. Later in this chapter a feedback based scheme will be introduced for online adjustment that enforces the target lifetime in spite of the errors.

5.2 Policies for Battery Discharge Rate Control

The previous section discusses the problem of determining the appropriate discharge rate for target lifetime. This section introduces a set of currentcy based policies to control the battery discharge rate to the desired level. In the energy management framework, there can be multiple possible policies for a single function module. The policies introduced in this chapter are the set of policies for the specific goal of this chapter and for the simplified scenario of one running application. They are subject to enhancement or revision when more goals are proposed or a more general scenario is considered.

5.2.1 Distribution Policy

The distribution policy plans the energy consumption over time. As suggested in equation 5.3, the current distribution policy divides the available energy evenly along the lifetime so that the system has constant power supply. Every time the distribution module is invoked, it first checks the battery status and the remaining lifetime, and then calculate the $AverageDischargeRate$ according to equation 5.3. The $AverageDischargeRate$ is used to generate currentcy every epoch, according to the equation below:

$$Currentcy_{total} = (AverageDischargeRate - BasePower) * EpochLength * JtC \quad (5.4)$$

The JtC in the equation is used to convert the unit of $Joule$ to the unit of $currentcy$. It equals to 100000 in the current hardware platform. The epoch length is empirically set to 1 second for a trade-off between allocation overhead and smooth execution.

Unlike real-time scheduling where future activities are known a prior, or a scenario where application hints are available, ECOSystem assumes a scenario of running unmodified general purpose applications which may change their behavior dynamically. The approach of spreading available battery energy over lifetime can also be viewed as reserving

energy proportional to the remaining lifetime for later events or newly started applications. However for an application with a fixed amount of computation, this policy will cause the execution of this application to be simply stretched, especially in the scenario of one running application. The performance impact of the distribution policy is shown in section 5.5.

5.2.2 Allocation Policy

With the assumption of one running application, the allocation policy can simply dump all currentcy into the container of the only application. In ECOSystem, every container has a cap to limit the amount of accumulated currentcy. Container cap is used to allow the application to accumulate currentcy for expensive operations but also limit the application's bursty behavior. Based on these objectives of the container cap, the allocation policy sets the cap proportionally to the currentcy allocation every epoch:

$$Cap = N * Currentcy_{total}, N = 10 \tag{5.5}$$

According to this equation, starting with 0 currentcy, a task could accumulate currentcy for 10 seconds before the cap is reached. All currentcy exceeding the cap is simply discarded.

When currency is discarded, the energy itself is not wasted. The energy is still in the battery. Discarding currentcy will result in decreased battery energy consumption, and according to equations 5.3 and 5.4, it will lead to an increase in the discharge rate and currentcy generation in later epochs.

5.2.3 Scheduling Policy

Traditionally, with one application, the application is simply granted any available resource it demands. With the energy constraint, the application execution has to be blocked even

when the resource is available. With only one application, the scheduling considered in this chapter does not schedule the application against other competing applications to share the resource. Rather it can be viewed as scheduling the application against an idle process to throttle the application's energy consumption rate.

The currentcy based energy management framework already has the epoch based mechanism to throttle the energy consumption rate, thus there is no need to implement another independent scheduler for energy consumption. This epoch based "pay-as-you-go" (PAYG) policy does not modify the default Linux scheduler, instead having currentcy in the container becomes the necessary condition for an application to be scheduled.

5.3 Feedback Based Adjustment

Thus far many components of ECOSystem has been described that work together for the goal of achieving target lifetime. However, as in all systems, errors can occur in these components that deviate the system from the goal. This section introduces an online adjustment module that keeps the system on track with the goal. The online adjustment module is based on control theory. This section first briefly introduces the control theory and how it can be applied to ECOSystem. Control theory is then used to analyze ECOSystem, and finally a feedback module is proposed for online adjustment.

5.3.1 Brief Introduction to Control Theory

Control theory studies the dynamics of a system and how it responds to external signals. It is widely applied in many areas, from spaceship to micro-chips, from economic policy-making to neural networks. For instance, in the design of TCP/IP, control theory is used to analyze how the protocol responds to the changing environment, such as decreased bandwidth. ECOSystem has to be adaptive as well, such as to the dynamically changing behavior of applications or the inaccuracy in hardware energy consumption modeling. Control

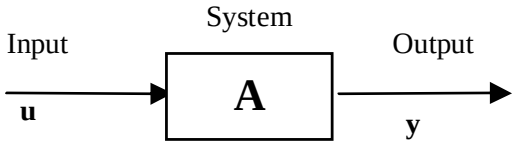


Figure 5.3: Input-output Block Diagram

theory can be used in ECOSystem to analyze how the system responds to these changes and how the system can be more robust.

Figure 5.3 shows the block diagram of a system with one input u and one output y. A system can be classified as either a static system or a dynamic system. A static system directly maps the input to the output, described by the algebraic equation:

$$y = \mathbf{A}u \tag{5.6}$$

u is the input variable, y is the output variable and $\mathbf{A}$ is the transformation variable between input and output. Control theory studies dynamic systems which are described using differential/difference equations for continuous/discrete systems. The class of dynamic systems we are interested in is the *linear time invariant dynamic systems*. A general form of a discrete linear time invariant dynamic system can be described as:

$$\begin{aligned} &y(k+n) + a_{n-1}y(k+n-1) + ... + a_1y(k+1) + a_0y(k) = \\ &\quad b_mu(k+m) + b_{m-1}u(k+m-1) + ... + b_1u(k+1) + b_0u(k) \end{aligned} \tag{5.7}$$

In the equation 5.7, all coefficients a_i and b_j are constant. As can be seen from the equation, for a dynamic system, the output is not only determined by the current value of the input, but also the past values of the input. This equation only describes a system with one input and one output. Systems with multiple inputs and outputs can be described using transformation matrix and input output vectors.

With the equation 5.7, the system is described mathematically in the time domain. Although it is possible to directly analyze the system described with discrete functions, the analysis is usually done in the $\mathcal{Z}$ domain through $\mathcal{Z}$ transformation. With the $\mathcal{Z}$ transformation, the system is described in the frequency domain. Even though a system can be described in either domain, system dynamics are usually more clearly expressed in the $\mathcal{Z}$ domain. After the transformation,the difference equation 5.7 can be turned into a polynomial form:

$$(z^n + a_{n-1}z^{n-1} + ... + a_1 z + a_0) * y(z) = (b_m z^m + b_{m-1}z^{m+1} + ... + b_1 z + b_0) * u(z) \quad (5.8)$$

One important function of a control system is the transfer function that shows the relationship between input and output:

$$G(z) = \frac{y(z)}{u(z)} = \frac{z^n + a_{n-1}z^{n-1} + ... + a_1 z + a_0}{b_m z^m + b_{m-1}z^{m+1} + ... + b_1 z + b_0} \quad (5.9)$$

The transfer function represents the characteristic of the system which is independent of the input. Given a certain input, the output can be calculated as: $y(z) = G(z) * u(z)$. Introduction to control theory can be easily found elsewhere. Here it is assumed that the reader already has some background knowledge about control theory.

5.3.2 ECOSystem Analysis

In this section, control theory is used to analyze the effect of errors in the energy consumption modeling, for instance if the 15.5W CPU is incorrectly modeled to be 12W. It is assumed that there is one device incorrectly modeled and the application spends a fixed portion of its currentcy allocation on the device. Errors of multiple devices can be weighted into a single device and have the same analysis.

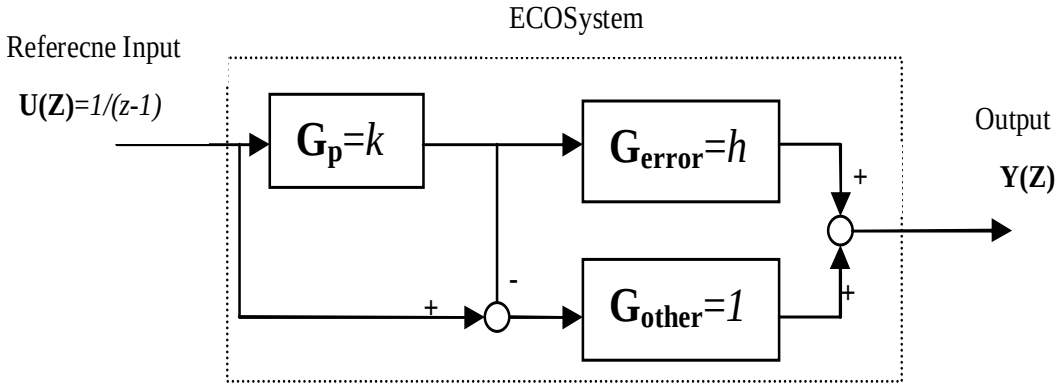


Figure 5.4: ECOSystem System Block Diagram

For ECOSystem, there are two types of analysis: the system stability analysis and steady state error analysis. A system is called $stable$ if the system's motion is bounded. A system is $asymptotically\ stable$ if in addition to being stable, the system's motion will eventually converge to a steady state. An unstable system can not work properly. In this section it is interesting to see if ECOSystem is stable with the presence of errors in energy consumption modeling.

For an asymptotically stable system, the steady state error analysis means that for a given input, if there is a difference between the actual output and the desired output when the system is in the steady state. The steady state error analysis is applied to ECOSystem to see how the error affects the system output.

ECOSystem can be model as an open-loop system shown in figure 5.4. The input $U(Z)$ is the desired power consumption level according to the target lifetime. The output $Y(Z)$ is the actual system power consumption. $U(Z)$ is also called $reference\ input$ as the goal of ECOSystem is to match the output $Y(Z)$ to the input.

The reference input $U(Z)$ is modeled with a step function, $U(Z) = \frac{1}{Z-1}$ represented in the $\mathcal{Z}$ domain. ECOSystem generates a certain amount of currentcy based on the input $U(Z)$. A fixed portion of the currentcy generated goes to the incorrectly modeled device,

$G_p(Z) = k, 0 \leq k \leq 1$ is the portion of total currency. The device then transforms the currentcy spent on it into energy consumption, $G_{error}(Z) = h, 0 \leq h$, h is the ratio between the actual power consumption and modeled power consumption. For instance, if the 15.5W CPU is modeled to be 12W, then $h = 15.5W/12W = 1.29$. The application spends $1 - G_p$ of its currentcy on other devices that transform the currency into the same amount of energy, $G_{other}(Z) = 1$.

System stability is strictly related to the system transfer function. According to the block diagram, ECOSystem has the following equation

$$U * G_p * G_{error} + U * (1 - G_p) * G_{other} = Y \tag{5.10}$$

Thus, the transfer function between $U(Z)$ and $Y(Z)$ is

$$G(Z) = \frac{Y(Z)}{U(Z)} = G_{error} + (1 - Gp)G_{other} = kh + (1 - k) = (1 - k + kh) \tag{5.11}$$

Here another constant $g = 1 - k + kh$ is introduced, which is the static overall gain of the system. When applying stability analysis on the transfer function $G(Z) = g$, because it does not has any pole, the system is always asymptotically stable.

The steady state error of the system is the difference between $U(Z)$ and $Y(Z)$.

$$E(Z) = U(Z) - Y(Z) = U(Z)(1 - G(Z)) = \frac{k(1 - h)}{Z - 1} \tag{5.12}$$

According to the final value theorem, the value of $e(x)$ ($E(Z)$ is the $(\mathcal{Z})$ transform of $e(x)$) as $n \to \infty$ is

$$\lim_{x \to \infty} e(x) = \lim_{Z \to 1}((Z - 1)E(Z)) = k(1 - h) \tag{5.13}$$

According to equation 5.13, the steady state error of ECOSystem does not equal to zero when $h \neq 1$, which quite matches the intuition.

To sum up, the modeling error will not cause ECOSystem to become unstable, but it will cause steady state error in the actual energy consumption of ECOSystem.

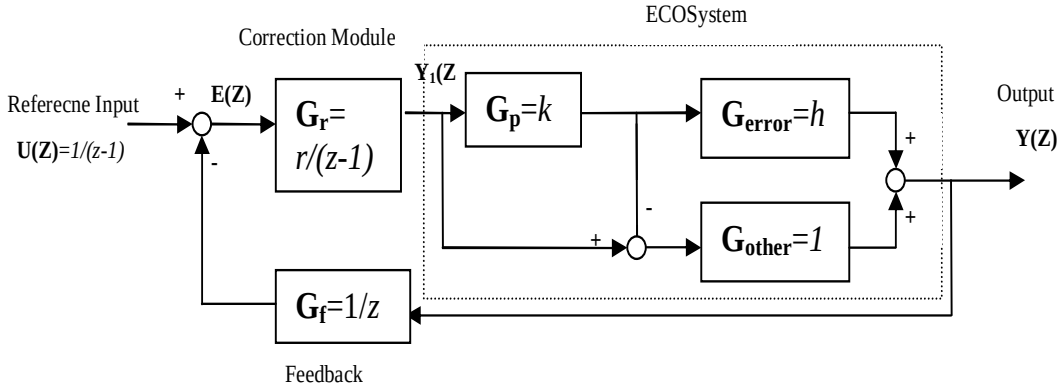


Figure 5.5: ECOSystem with Feedback Based Control

5.4 Feedback Control Module

One way to eliminate the steady state error is to add a feedback control module into the system, as shown in figure 5.5. The output $Y(Z)$ is fed back to the feedback module and compared to the reference input $U(Z)$, $E(Z) = U(Z) - Y(Z)$. When there is a difference between these two, the feedback control module can try to eliminate the error by adjusting the amount of total currentcy allocated each epoch. For instance when the system is consuming more energy than desired, the feedback module can decrease the total currentcy allocation continuously to throttle energy consumption until $Y(Z) = U(Z)$. When the feedback control module is on, the equation 5.3 used to determine the currentcy allocation level each epoch is replaced.

The feedback module is not turned on when the consumed energy is less than the target level and the application is not throttled, in which case it is not necessary to change the currentcy allocation. The feedback module adjusts the currentcy allocation level and approaches the reference input with small steps. Every step only a small adjustment is made. The algorithm is described in table 5.4.

The $TargetEnergyConsumption$ corresponds to the reference input $U(Z)$. The $EnergyConsumption_{n-1}$ is the output $Y(Z)$, the actual energy consumption, measured

$$
\begin{aligned}
&E_n = EnergyConsumption_{n-1} - TargetEnergyConsumption \\
&Delta = 0.2 * E_n \\
&TotalCurrentcy_n = TotalCurrentcy_{n-1} + Delta
\end{aligned}
$$

Table 5.1: Feedback Control Algorithm

by monitoring the status of the battery. The output is fed back by the module $G_f(Z)$. Because the energy consumption of the last step is used, $G_f(Z)$ has the transfer function $G_f(Z) = 1/Z$. The difference between these two, E_n, is the only input to the feedback control module. During the adjustment, instead of directly using E_n, the next currentcy allocation is only adjusted by a small amount $0.2 * E_{n-1}$. This is used to prevent correction overshoot. The value of 0.2 is determined empirically, its effect on the system stability will be shown in the analysis below.

According to the algorithm described in table 5.4, the controller has the equation $ZY_1(Z) = Y_1(Z) + rE(Z)$, $Y_1(Z)$ is the output of the controller. The transfer function of the controller should be $G_r(z) = \frac{Y_1(Z)}{E(Z)} = \frac{r}{z-1}$, $r = 0.2$. The time interval between steps is 10 seconds for a tade-off between overhead and timely response.

According to the block diagram in figure 5.5, we have the following equations

$$
\begin{aligned}
&E = U - Y * G_f \\
&Y = E * G_r(G_p * G_{error} + (1 - G_p) * G_{other})
\end{aligned}
\tag{5.14}
$$

From equation 5.14, the transfer function of the new system is

$$
G(z) = \frac{Y}{U} = \frac{G_r * G_p * G_{error} + G_r * (1 - G_p) * G_{other}}{1 + G_f * G_r * G_p * G_{error} + G_f * G_r * (1 - G_p) * G_{other}} = \frac{rg}{Z^2 - Z + rg}
\tag{5.15}
$$

In the equation $g = 1-k+kh$ is the overall gain of the ECOSystem, caused by the incorrect energy consumption modeling. According to control theory, in order for the system to be

stable, the two roots of the equation $Z^2 - Z + rg = 0$ must lie within the unit circle of the $\mathcal{Z}-$plane, or

$$\begin{array}{l} |1 + \sqrt{1 - 4rg}| < 2 \\ |1 - \sqrt{1 - 4rg}| < 2 \\ r > 0, g > 0 \end{array} \tag{5.16}$$

Solving the inequalities above can come to the conclusion that the system is asymptotically stable if and only if $1 > rg$. The system may become unstable because the system itself has an amplifying effect if $g > 1$. With the introduction of the feedback control module, the small adjustment $rE(Z)$ can be greatly amplified and cause over-shooting and system oscillation. When r equals 0.2, the system will be unstable if $g > 10$, or in the previous example the 15.5W CPU is incorrectly modeled to be 1.5W (modeling error of an order of magnitude).

Controller design needs to target to the dynamics of the system. The challenge with designing a control module for ECOSystem is that the dynamics of the system (the gain of the system to be more specific) is unknown. ECOSystem takes the approach of fine-tuning the parameter of r so that the system instability can rarely happen in practice. Designing an always stable controller requires advance knowledge of control theory, and is planned for the future work.

When the system is asymptotically stable, the steady state error is

$$E(Z) = U(Z) - Y(Z) = U(Z)(1 - G(Z)) = \frac{1}{Z - 1} * \frac{Z^2 - Z}{Z^2 - Z + rg} \tag{5.17}$$

And according to the final value theorem

$$\lim_{x \to \infty} e(x) = \lim_{Z \to 1}((Z - 1)E(Z)) = 0 \tag{5.18}$$

With the feedback controller, the steady state error of the system is zero if and only if the system is asymptotically stable, or $1 > rg$.

Target Battery Lifetime (hour)	1.5	1.75	2.0	2.25	2.5
Achieved Battery Lifetime (hour)	1.48	1.75	1.99	2.23	2.45

Table 5.2: Achieving Target Battery Lifetime

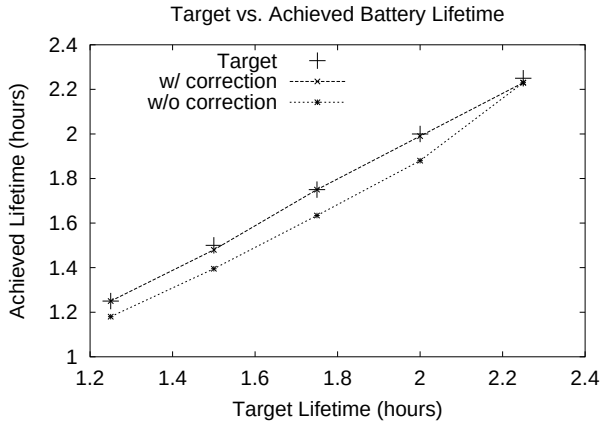


Figure 5.6: Battery Lifetime with Feedback Control

The conclusion with control theory analysis is that ECOSystem may become unstable with the introduction of the feedback control module, although in practice the possibility is very low. But when ECOSystem is stable, the steady state error of the system is reduced to zero.

5.5 Battery Lifetime Experiments and Results

This section evaluates the currentcy based energy management with the goal of achieving a target battery lifetime. The first subsection experiments on ECOSystem's ability to achieve this goal. This is followed by the experiments on its impact on applications' performance.

5.5.1 Achieving Target Battery Lifetime

To evaluate ECOSystem's ability to manage energy resource, one CPU intensive microbenchmark is selected to run on ECOSystem and the target lifetime is set with different values. In

this experiment, tests are always started with 3.4Ah energy remaining in the battery. When running unthrottled, this microbenchmark consumes more than 15W on the CPU. Combined with the base power consumption of about 13W, the battery can last for about 1.2 hour without energy management. With energy management, ECOSystem can achieve a maximum battery lifetime of about 2.6 hours with only the base power. Any lifetime goal above 2.6 is rejected by ECOSystem as unachievable. A platform of lower base power value would allow more flexibility in choosing the target battery lifetime.

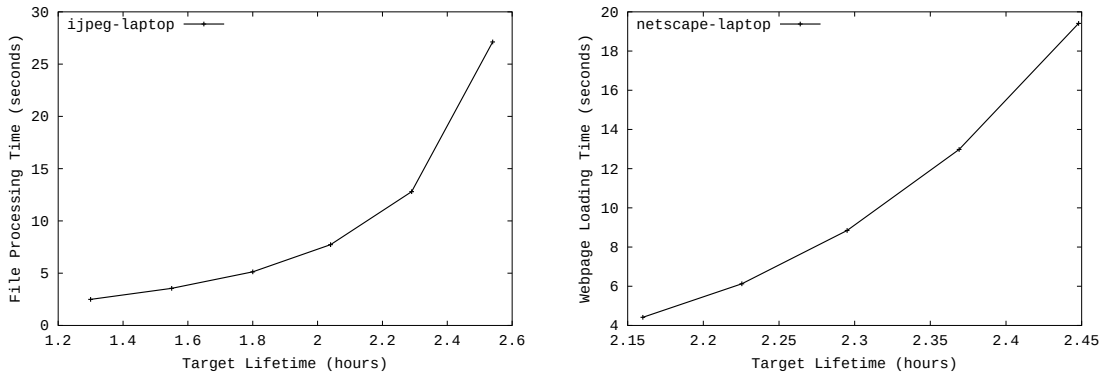
This experiment sets the target lifetime from 1.5 hour to 2.5 hour. As shown in the table 5.2, the achieved target battery lifetime closely matches the target value even without any feedback control, which shows the effectiveness of the energy management framework.

One advantage in using a microbenchmark is that its activity is regular and its energy consumption on the CPU is modeled pretty accurately. For real applications several potential sources of error may exist. For example, variations in cache behavior that are not captured by the flat CPU charge may introduce error in the lifetime estimate. One remedial approach that has been investigated involves making periodic corrections with the feedback control module. By regularly obtaining the remaining battery capacity via the smart battery interface, ECOSystem can take corrective action by changing the amount of currentcy allocated in subsequent energy epochs. If the system appears to be under charging, then the overall currentcy allocation can be reduced. If it appears that the system is over charging, then currentcy allocation can be increased.

To investigate the impact of energy accounting inaccuracies, the same CPU intensive microbenchmark is used, but accounting error is deliberately introduced for the CPU power consumption (14W instead of the measured 15.55W). Figure 5.6 shows the target battery lifetime on the x-axis and the achieved battery lifetime on the y-axis. One set of points is plotted along the $y = x$ line that represents perfectly matching the lifetime. One curve demonstrates the behavior of the system without correction, in this case continuously miss-

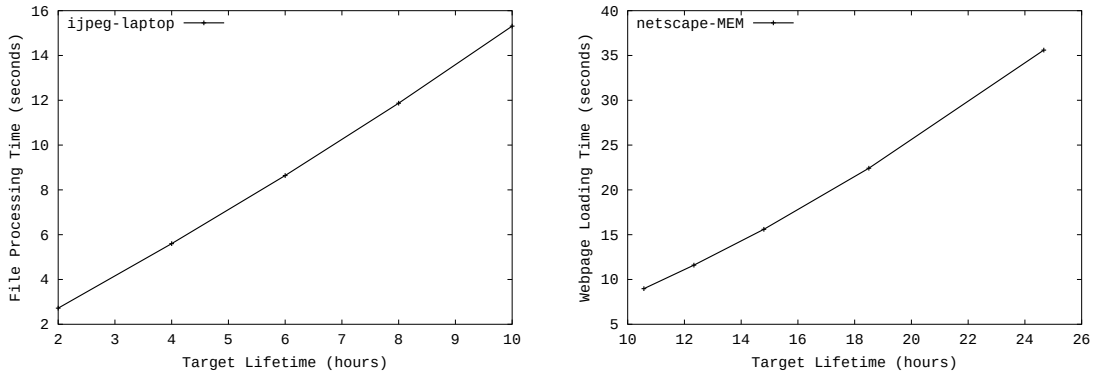
ing the target battery lifetime by approximately 10%. Finally, another curve on the graph shows that with the periodic corrections, ECOSystem is able to achieve the target despite the deliberately introduced error.

5.5.2 Application Performance Impact



a) Ijpeg b) Netscape

Figure 5.7: Application's Performance vs. Battery Lifetime



a) Ijpeg b) Netscape

Figure 5.8: Application's Performance vs. Battery Lifetime: Alternative Platform

This subsection studies the application's performance impact from power consumption regulation. Please note that at this stage ECOSystem focuses on the energy management

framework for global energy-related goals. Improving application's performance within this framework is discussed in chapter 8. The goal of this experiment, however, is to investigate how the application's performance degrades with limited energy resource, and how the performance is affected by hardware power characteristics.

Figure 5.7-a shows the ijpeg's performance with different battery lifetime, with $ijpeg$ being the only application in the system. The most important thing conveyed by the graph is that the file processing delay increases dramatically with the battery lifetime; The processing delay is more than tripled when the lifetime increases from 2.04 hour to 2.54 hour. This is because the base power consumes energy consistently. With the increase of battery lifetime, the portion of total energy consumed by the base power increases proportionally, thus less energy is left for the application to do useful work. The same phenomenon is also shown in figure 5.7-b by $netscape$. $Netscape$ has a smoother curve because currentcy can be accumulated during the think time.

To validate the hypothesis for the phenomenon of figure 5.7, it would be helpful to experiment with these applications on another hardware platform of totally different power characteristics. For ECOSystem, this can be done by utilizing the flexibility of the model to emulate an entirely different platform.

Such an emulation is most accurate when the power characteristics change without significant timing changes (e.g., as lower power processors are introduced that are capable of matching the host platform's speed). Recognizing this limitation, ECOSystem can emulate a hypothetical platform that is representative of a future PDA-like device with a 2W processor, 0.02W base power, and MEMS-based storage. The MEMS storage power characteristics are based on those presented by Schlosser et al [47]. An access costs 0.112mJ, transitioning to active mode costs 5mJ and transitioning back to standby mode costs 0mJ. It is assumed that the energy to remain active is 100mW, and the timeout to standby mode is 50ms. The same payback policy is used as in the IBM hard disk model. For these

experiments a 3.7Ah battery with 1.5v operating voltage is assumed.

Figure 5.8 shows the result of the same two applications on this alternative platform. One thing can be noticed is that with the base power consumption of only 0.02W, battery lifetime can be managed within a much wider range. Theoretically it can be any number between 2.75 hour to 277 hour. The next thing can be noticed is that the delay of both applications now increase linearly with the lifetime on the alternative platform. Note that this alternative platform is only used in this experiment. All other experiments in this thesis are performed on the laptop platform.

This experiment shows how the currentcy model can be modified to support alternative platforms by simply changing device parameters. An important observation from these results is that improvements in device power consumption increase both the need and the opportunity for operating system managed energy.

5.6 Summary

In order to focus on the problem of battery management, this chapter assumes a scenario of only one application to simplify the problem of resource sharing and scheduling. The first step of battery management is to estimate the amount of remaining energy that can be extracted from the battery. However, this information can not be easily acquired because of battery's non-linear electro-chemical properties. The approach used in ECOSystem is utilizing the existing battery interface for such estimation. Current battery is capable of monitoring the battery status itself and processing inquiries inside the battery, which relieves the OS from the overhead and hardware details. But with this approach, current ECOSystem is limited by the simple Control Method battery interface of the platform and only has a very simple estimation of the remaining battery energy.

The second step of battery management is to control the battery drain rate to the desired level. This is realized by the currentcy model and the management framework. To deal

with errors in the device energy consumption modeling that may deviate the system from the goal of achieving a target battery lifetime, ECOSystem incorporates a feedback-based module for online correction. This module monitors the actual drain rate of the battery and compares it to the desired level. If there is a difference between these two, correction can be made by adjusting the total currentcy allocation each epoch. The auto-correction module is based on control theory. A brief introduction to the control theory and an analysis of the auto-correction module is presented in this chapter.

In the experiment section, it is shown that when devices are modeled accurately, the management framework is able to achieve the target battery lifetime. In another example, it is shown that with errors in the system, the auto-correction module can enforce the desired battery drain rate and the battery lifetime can still be achieved.

Chapter 6

Reducing Residual Battery Energy with Currentcy Conserving Allocation

In the previous chapter, battery management is studied in the scenario of only one task. This chapter moves on to the battery management in the general scenario of multiple concurrent tasks. One difference of the new scenario is that currentcy needs to be shared among tasks by the allocation module. The allocation policy is also shown to be able to affect the residual energy in the battery. The first section introduces the goal of reducing residual battery energy and how it is affected by currentcy allocation. Then the currentcy conserving allocation policy is proposed that reclaims surplus currentcy to reduce residual battery energy.

6.1 Allocation Policy and Residual Battery Energy

The previous chapter focuses on the goal of target battery lifetime. Another implicit goal considered in the battery management is to reduce residual battery energy. The section discusses this goal in the multiple task scenario and shows how it is related to the currentcy allocation policy.

6.1.1 Reducing Residual Battery Energy

Residual battery energy is the leftover energy remaining in the battery when the target lifetime is reached. The goal of reducing residual battery energy is to utilize the battery more efficiently because residual energy usually means lost opportunity for tasks to do more work when energy is the constraint.

This goal and the goal of target lifetime complement each other in the management of the battery: the energy consumption rate should be throttled so that the system can function throughout the entire lifetime, at the same time as much energy should be extracted from the battery as possible to power more task activities.

6.1.2 Battery Management for Multiple Tasks

When managing the battery, ECOSystem tries to maintain an appropriate power level that depletes the battery right when the target lifetime is reached. To ensure that the system is not over-consuming nor under-consuming energy, the feedback control module monitors the battery status and adjusts the overall currentcy allocation. This mechanism should work equally for both single task scenario and multiple tasks scenario, as can be shown by experiments in section 6.6.

One exception for the battery management mechanism is when a task can not consume the allocated currentcy, in which case the actual energy consumption can no longer be controlled by the currentcy allocation. According to the allocation policy described in the previous chapter, surplus currentcy that exceeds the container cap is simply discarded. For the one task scenario, discarding currentcy does not cause wasted energy. The energy is still in the battery to be consumed by the same task.

In the multiple tasks scenario, it is possible that some of the tasks have surplus currentcy while other tasks are throttled by the limited currentcy allocation. This can happen when one critical task is given a large currentcy allocation to guarantee its worst case currentcy demand while most of the time the task can only consume a portion of the allocation. The policy regarding the treatment of surplus currentcy can affect the residual battery energy, which is discussed in the next subsection.

6.1.3 Impact of Allocation Policy on Residual Energy

There are many possible policies on how to deal with the surplus currentcy. One policy is to allow currentcy be accumulated without limit. Although it seems to be "fair" that a task does not lose its allocated currentcy, it may increase the residual battery energy because the accumulated currentcy may never be spent.

Task	Power Consumption Unthrottled	Allocated Power	Surplus Power
Task A	7,000mW	8,000mW	1,000mW
Task B	15,500mW	4,000mW	0mW

Table 6.1: Example of unlimited accumulation allocation policy

Table 6.1 shows an example of this *unlimited accumulation* policy. In this example task A's power consumption ranges between 0 and 15,500mW, but on average it consumes 7,000mW when running unthrottled. Task B consumes 15,500mW running unthrottled. In this example the system can allocate a certain amount of overall currentcy every epoch that corresponds to the power consumption level of 12,000mW. Of the total power available, task A is allocated 8,000mW and task B is allocated 4,000mW. According to the $unlimited\ accumulation$ policy, the 1,000mW surplus power of task A will remain in its container in the form of currentcy and never be consumed, generating a residual energy of 8.3% of original battery capacity.

The policy used in the previous chapter simply discards the surplus currentcy of task A. This policy is now referred to as the $piggy\ bank$ policy because the revoked currentcy essentially goes back to the battery and will be re-allocated to both task A and B in later epochs because of the constant battery monitoring of the feedback module.

The problem with the $piggy\ bank$ policy is that tasks can not get steady currentcy allocation during the lifetime. Also this policy is too conservative in that later epochs always get more currentcy allocation than earlier epochs. Experiments in section 6.6 show

that this policy can also cause large residual energy because tasks may not able to consume the currentcy saved up in the $piggy\ bank$ when the target lifetime is reached.

One alternative policy is to give the surplus currentcy of task A to task B to maintain a steady stream of currentcy allocation. This policy is referred to as the $currentcy\ conserving$ policy. There are some questions still remaining to be answered for this policy: How to re-allocate the surplus currentcy if it is needed by several tasks? What happens if one of the tasks reaches the container cap during the re-allocation? How to do the re-allocation more efficiently?

This section briefly describes three allocation polices. Through one example, it is shown that the surplus currentcy reclamation of allocation policy does have an impact on the residual battery energy. Currentcy conserving allocation will be further discussed in the following sections.

6.2 Currentcy Reclamation in ECOSystem

This section discusses the ECOSystem currentcy allocation and reclamation in the multiple tasks scenario. The first subsection introduces how the resource is shared in ECOSystem.

6.3 Resource Sharing Abstraction in ECOSystem

When there are multiple tasks, an abstraction is needed to specify the relative importance of tasks and how the resource is shard according to their importance. In ECOSystem, every task is assigned a $share$ value and ECOSystem allocates currentcy to tasks proportional to their $share$. The currentcy allocation for task i is calculated as

$$CurEntitled_i = Currentcy_{total} * \frac{share_i}{\sum_{k=1}^{n} share_k} \tag{6.1}$$

In the equation, $Currentcy_{total}$ is the overall currentcy allocation for all tasks in the epoch. $Share_i$ is the $share$ of task i and n is the total number of tasks in the system. In this section the discussion of resource sharing abstraction focuses on how it is used in currentcy allocation. The resource abstraction for providing differentiated service is discussed in the next chapter.

The $share$ based proportional allocation is very similar to the resource sharing abstractions in other systems. In [60] every network connection is assigned a $weight$ and share the network bandwidth proportionally. In Linux, every process has a $nice$ value that can be translated into a certain portion of CPU time. [55] uses $ticket$ as the abstraction for resource sharing. Resources are allocated to task proportional to their number of $tickets$. There are some advanced features for the $tickets$ abstraction, such as transferring $tickets$ from one task to another. Also the system can issue $hard\ tickets$ to a task that represents a fixed amount of resource instead of a relative share. These features are not implemented in the current ECOSystem, but they can be easily incorporated.

6.3.1 Resource Reclamation

The currentcy allocation in equation 6.1 is referred to as the $entitled$ currentcy because its calculation is purely based on the $share$ value assigned to the task, without considering the task's currentcy demand. The $entitled$ currentcy may not equal to the actual currentcy consumption, rather it is the guaranteed currentcy allocation when it is needed by the task. The surplus currentcy can be seen as the difference between $entitled$ currency and actual currentcy consumption.

The surplus currentcy, although a part of the entitled currentcy, needs to be reclaimed by the *currentcy conserving* allocation for efficient use of the resource. This process also exists in the management of other resources. In the $stride$ CPU scheduling [55], if a task has one half of the total tickets in the system, it only gets the entitled CPU time when

it is always in the run queue ready for CPU. When it is not ready, its otherwise idled CPU time will be shared by other ready tasks. In the network packet scheduling, even if a connection has reserved a certain amount of bandwidth, the scheduler will not let the bandwidth be wasted when there is no packet of the connection in the input queue [60]. [37] contains a definition for *weighted max-min fairness* that defines the treatment of the surplus bandwidth resource.

As shown in the discussion, resource reclamation has several parts. First the entitled resource should be calculated, then the actual resource demand identified, and finally the surplus resource be re-allocated. In the CPU scheduling or packet scheduling, the demand for the resource can easily be identified by whether the task or packet is in the queue. For currentcy allocation, this is not so obvious as currentcy can be deducted by various devices, and it can be deducted asynchronously with task activities. For the CPU or bandwidth resource, the surplus resource is re-allocated by excluding the unready tasks from the resource allocation. For the currentcy resource there is no such concept of ready task set because even idle tasks may still need currentcy.

6.4 Identifying Currentcy Consumption Ability

With currentcy conserving allocation, the actual currentcy consumption can be quite different from the entitled currentcy. In this section, it is proposed that the container cap should be set adaptively based on the the actual currentcy consumption instead of the $share$ value, so that the amount of the currency in the container can be a real indication of task's currentcy consumption ability.

6.4.1 Container Cap and Currentcy Consumption

In ECOsystem, the container cap is used as the threshold to indicate whether or not the task can receive any more currentcy. In the initial design of the mechanism of the previous

chapter, the cap of container is set proportional to the entitled currentcy. The multi-task version of the cap setting algorithm of equation 5.5 is

$$Cap_i = N * CurEntitled_i,\ N = 10 \tag{6.2}$$

$CurEntitled_i$ is the entitled currentcy for task i calculated by equation 6.1.

On the other hand the currentcy consumption of a task may also be affected by the cap setting algorithm. Table 6.2 shows an example of currentcy allocation within an epoch. In this example, task $A \sim D$ are identical tasks that each has 3,000 surplus currentcy every epoch. Task E is the only task in this example that can use the surplus currentcy. It can consume 15,000 currentcy every epoch but only has the entitled allocation of 1,000 currentcy. Its cap is set to 10,000 currentcy according to equation 6.2.

Tasks with Surplus Currentcy	**Entitled Currentcy**	**Container Cap**	**Surplus Currentcy Generated**	**Currentcy Consumed**
Task A	8,000	80,000	3,000	5,000
Task B	8,000	80,000	3,000	5,000
Task C	8,000	80,000	3,000	5,000
Task D	8,000	80,000	3,000	5,000
Task limited by Currentcy	**Entitled Currentcy**	**Container Cap**	**Surplus Currentcy Received**	**Currentcy Consumed**
Task E	1,000	10,000	9,000	10,000

Table 6.2: Allocation Policy Example

With surplus currentcy reclamation, a task can receive currentcy more than its entitled share. Although the exact currentcy conserving allocation algorithm is not introduced yet, in this example it would be more efficient if the 12,000 surplus currentcy from task $A \sim D$ was given to task E. However with the currentcy cap of 10,000, only 9,000 of the surplus currentcy can be put into the container. The container cap of task E should be increased to accommodate more surplus currentcy. Similarly, for task $A \sim D$ that consistently spend only a fraction of their entitled energy, the cap can be decreased to free more currentcy from the container. This example shows that the cap setting of a container should be based on

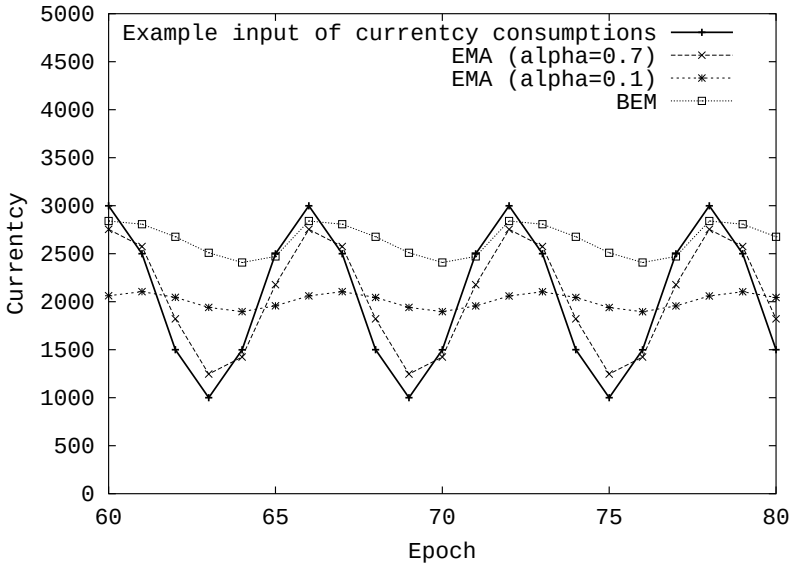


Figure 6.1: Filters for tracking currentcy consumption

the actual currentcy consumption, which could be quite different from its entitled currentcy allocation.

6.4.2 History Based Currentcy Consumption Estimation

The actual currentcy consumption of a task is determined by many factors, such as its own running characteristics, resource constraints, data or user input, and competition from other tasks. In ECOSystem a task's future currentcy consumption is predicted by looking at its historical information

$$CurAverage_{i,n} = \alpha * CurConsumption_{i,n} + (1 - \alpha) * CurAverage_{i,n-1}, \quad 0 < \alpha < 1 \tag{6.3}$$

Equation 6.3 is an exponential moving average (EMA) filter that tracks the average of previous currentcy consumptions. $CurAverage_{i,n}$ is the EMA output for task i of epoch n, it is a weighted sum of the currentcy consumption of the epoch and the EMA output of the last epoch. α is the relative weight of the two components in the calculation. A large α places more emphasis on the most recent currentcy consumption, thus the $CurAverage$

will respond to currentcy consumption changes more rapidly. A small α value on the other hand produces a smoother average of previous currentcy consumptions. Figure 6.1 shows how two different α values, 0.1 and 0.7, respond to a example input.

In equation 6.3 the calculation of $CurAverage$ is affected by its values of previous epochs. It can also be viewed as a dynamic system with the currentcy consumption as the reference input and average as the output. According to equation 6.3, a system transfer function can be established: $G(Z) = \frac{1-\alpha}{Z-\alpha}$. Based on the transfer function, it can be easily concluded that the system is always asymptotically stable and it has zero steady state error for the input of step function.

6.4.3 Container Cap Setting

No matter whether the α is large or small, equation 6.3 tracks the average currentcy consumption, which may be higher or lower than the current value. The equation can not be directly used to set the container cap, because setting the cap too high or too low has different consequences. Setting the cap too high can cause more bursty behavior and more stockpiled currency. If the cap is set too low, the task may not be able to fully receive the surplus currentcy and cause residual energy, which is a much bigger problem.

Because of the different consequences, a filter is implemented that tracks the peak currentcy consumption instead of the average. The filter (BEM for Bi-weight Exponential Moving filter) uses two α values, α_h and α_l. The BEM filter and the cap setting algorithm is listed below.

$\alpha_h = 0.7, \alpha_l = 0.1, C = 5$
IF ($BEM_{i,n-1} \leq CurConsumption_{i,n-1}$) **THEN**
  $BEM_{i,n} = \alpha_h * CurConsumption_{i,n} + (1 - \alpha_h) * BEM_{i,n-1}$
ELSE
  $BEM_{i,n} = \alpha_l * CurConsumption_{i,n} + (1 - \alpha_l) * BEM_{i,n-1}$

$ContainerCap_{i,n} = (BEM_{i,n} + CurEntitled_i) * C$

Table 6.3: Container Cap Setting Algorithm

In the table 6.3, BEM tracks the peak currentcy consumption by going up quickly with α_h and going down slowly with α_l. Figure 6.1 shows how the BEM filter responds to the example input. To calculate the container cap, the $CurPeak$ is added to $CurEntitled$, and then multiplied by a constant to allow more currentcy accumulation.

6.5 Currentcy Conserving Allocation

This section introduces the final component of the allocation policy, the allocation algorithm. The first subsection gives the properties that define the currency conserving allocation, and the second subsection describes the allocation algorithm that can achieve these properties.

6.5.1 Currentcy Conserving Allocation Properties

$Currentcy\ conserving$ means that the overall currentcy for an epoch should be allocated to tasks as much as possible. During the allocation, each task has a $share$ that presents the task's priority for the resource. According to the definition of fairness in ECOSystem, both the allocation of entitled currentcy and the re-allocation of the surplus currentcy is proportional to the $share$ value. To be more formal, the currentcy conserving allocation needs to satisfy the following properties after the allocation for an epoch.

1. Conserving property: In an epoch, no currentcy is discarded unless all containers are full.

2. Proportional property: If there exist two containers that are not full after the allocation, their received currentcy in this epoch should be proportional to their $share$ value (the definition of fairness).

The currentcy conserving allocation only deals with the currentcy allocation within an epoch. The conserving property states that no currentcy should be wasted if it is needed by

a task. The second property states that unless the container is full, currentcy received by a task is proportional to its $share$. Note that the two properties include the allocation for both the entitled currencty and surplus currentcy.

6.5.2 Allocation Algorithm

There are many algorithms that can satisfy the two properties of currentcy conserving allocation. The algorithm used in ECOSystem is described here.

According to the two properties, for a total of n containers, if k ($k < n$) containers reach their caps during the allocation, the remaining currentcy will be allocated to the other $n - k$ containers proportional to their shares. The first step of our our allocation algorithm is to sort containers so that for any $container_i$ ($1 < i \leq n$), it will not reach its cap unless $container_{i-1}$ reaches its cap first. The rest of the algorithm simply walks through the sorted containers and allocates the currentcy in one pass. When $container_i$ is visited during the walk-through, because the allocation for all containers before $container_i$ is finalized, the remaining currentcy can be allocated to $container_i$ proportional to its $share$ among all $share$ values of remaining containers ($container_j, i \leq j \leq n$). The algorithm is described below.

1. Sort the containers in ascending order with the key value: $\frac{Capacity_i - CurAccumulated_i}{CurShare_i}$, $CurAccumlated_i$ is the remaining currentcy in the container before the allocation.

2. The variable $CurRemain$ is initialized to the total amount of currentcy for the epoch. Then containers are visited according to the sorted order. For each container, do the following.

3. Calculate the currentcy available to the container according to its share: $CurAvail_i = \frac{CurRemain * CurShare_i}{\sum_{j=i}^{n-1} CurShare_j}$

4. The currentcy available to the container is then truncated by the container cap: $CurAllocated_i = min(Capacity_i - CurAccumulated_i, CurAvail_i)$.

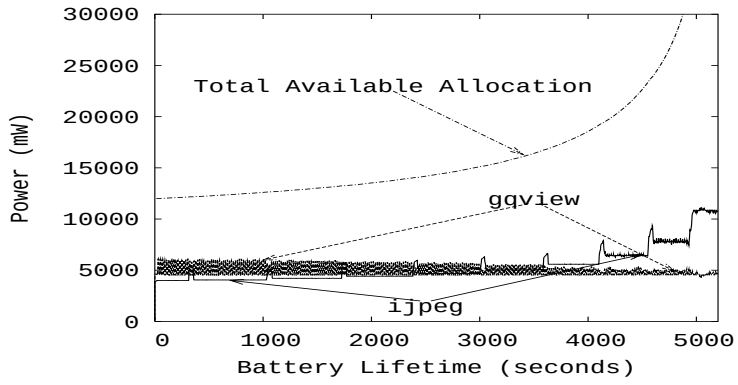
5. Currentcy is put into the container: $CurAccumulated_i = CurAccumulated_i + CurAllocated_i$

6. $CurRemain$ is updated for the next container ($container_{i+1}$): $CurRemain = CurRemain - CurAllocated_i$.

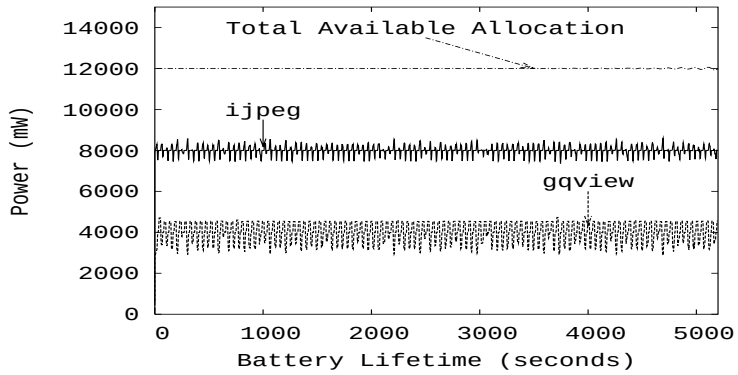
In the algorithm, sorting containers takes at most $O(n \log n)$ time, where n is the number of containers. The rest of the algorithm takes $O(n)$ to visit each container, so overall the algorithm takes at most $O(n \log n)$ time to finish.

6.6 Evaluation

This section validates the effectiveness of our currentcy conserving allocation in reducing residual battery energy. As a qualitative argument, please note that without an explicit energy-related abstraction similar to currentcy, it is difficult to articulate precisely what residual energy means or identify means to enforce a target battery lifetime. Monitoring the state of the battery as a separate device-specific resource offers little in the way of control over the resource. Thus, there is no "traditional" baseline policy with which it makes sense to compare. In this experiment, the currentcy conserving allocation is compared against the simple piggy bank currentcy allocation used in the previous chapter with its battery-level feedback mechanism that adaptively adjusts overall allocation levels. In the piggy bank policy, residual energy accumulates if a task does not spend all of its currentcy and has exceeded its currentcy cap causing that unspent currentcy to be lost. The experiment is designed to show that the piggy bank approach is less effective in reclaiming residual energy than explicit currentcy conservation.



a) Piggy Bank Allocation Policy



b) Currentcy Conserving Policy

Figure 6.2: Average power consumptions and total allocations for the Piggy Bank and the Currencty Conserving Allocation schemes

To evaluate the benefits of currentcy conservation, a workload consisting of the $gqview$ image viewer and $ijpeg$ is used. In this experiment, the target battery lifetime is set to 90 minutes, and the desired shares is set to be 66.6% for $gqview$ and 33.3% for $ijpeg$. These allocation settings correspond to an overall average power of 12,000mW with 8,000mW and 4,000mW for $gqview$ and $ijpeg$, respectively. This represents an overly generous allocation to $gqview$ which needs less than an average of 7000mW. $Ijpeg$, on the other hand, can easily consume up to 15.55W in the absence of other constraints.

Figure 6.2 shows the results. These plots show how the total allocation (presented in mW) changes over the lifetime of the battery. They also show the average power consump-

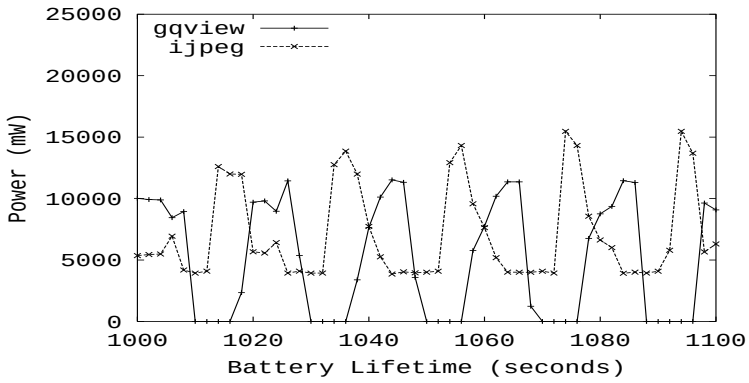


Figure 6.3: Time Varying Behavior with the Currencty Conserving Allocation

tion of the two applications for the three managed devices. The per-epoch measurements have been smoothed using a centered moving average over a window of twenty-one data points.

From these data, several observations can be made. First, for the original allocation policy (Figure 6.2a) it can be seen that the total power available for allocation (the top curve) increases dramatically near the end of the target battery lifetime. There is approximately 6.7% of the original battery capacity remaining at the end. The simple redistribution approach that returns gqview's unused currentcy (beyond the task's cap) to the overall energy resource initially spreads the excess over a large number of epochs, but as the target battery lifetime approaches there is less time over which to spread the excess. Intuitively, each epoch consumes only a fraction of the total excess and thus available energy continues to grow. In addition, gqview still receives its share of the increasing overall allocation that it does not need.

The second observation based on Figure 6.2a is that as time progresses, gqview's average power consumption (the middle line exhibiting some degree of scatter) decreases over time despite the increase in total availability. This is because the increase in available currentcy enables ijpeg (the bottom solid line that steps up toward the end of the lifetime) to consume more and more CPU time with the baseline CPU scheduling, undermining

gqview's ability to execute when it needs to in order to consume its currentcy.

Figure 6.2b shows the average power consumption of gqview and ijpeg when using the currentcy conserving allocation. It shows that there is no significant change in the available allocation as the two applications near the end of the target lifetime. Little residual energy capacity remains (less than 1%). By exploiting information in tasks' currentcy budgets, the currentcy conserving allocation policy successfully utilizes the available energy as compared to the approach that reacts to observed excess battery capacity.

There are variations in the power consumption of both applications due to gqview's execution variation that are not captured in the smoothed plots. To provide a better understanding of the simultaneous execution of ijpeg and gqview, Figure 6.3 presents the power consumption in each epoch over a 100 second time interval. This figure shows that when gqview is idle (i.e., during "think" time with zero power consumption) ijpeg can consume maximum CPU power. However, when gqview is active, ijpeg is limited to its 4000mW allocation. There is a brief delay in this transition that occurs while ijpeg's currentcy cap is adjusted. It can be observed that the power consumption of both gqview and ijpeg can exceed their allocation share because of currentcy accumulating up to their caps.

6.7 Summary

The general scenario of multiple tasks is assumed and the problem of efficient resource sharing is studied in this chapter. With an inefficient allocation policy, the target battery lifetime can be achieved, but may with large residual battery energy which means lost opportunity to do more work. To reduce the residual battery energy, unused resource of one task should be reclaimed and given to other tasks. In this chapter the currentcy conserving allocation is proposed that allocates as many currentcy as possible to tasks during an epoch.

For currentcy conserving allocation, the unused currentcy must first be identified. This is done in ECOSystem with the container cap. For each task, its container cap is adjusted

based on the currentcy consumption in previous epochs to reflect its currentcy consumption ability. Thus any surplus currentcy during the allocation that exceeds the cap is considered as unused resource for reclamation. Then an allocation algorithm is used in ECOSystem that allocates the currentcy, including the surplus currentcy from other tasks, to all tasks in the system. It is shown that with currentcy conserving allocation, the residual battery energy can be greatly reduced to less than 1% in our experiment.

Chapter 7

Differentiated Service with Energy Consumption Scheduling

The goal studied in this chapter is providing differentiated service to tasks. In ECOSystem, this means that critical tasks can consume more of the limited energy resource. However, it is shown in the previous chapter that simply allocating more currentcy to a critical task does not always result in increased energy consumption or performance. In this chapter, it is shown that in addition to differentiated allocation, scheduling support is also necessary for providing differentiated service. This chapter introduces the architecture for schedulers to be coordinated and consistent with the allocation policy. Two schedulers, an energy centric CPU scheduler and a network bandwidth scheduler, are implemented in ECOSystem, which are evaluated in the last section.

7.1 Priority Inversion in ECOSystem

The resource sharing abstraction $share$ has been introduced in the previous chapter for the allocation policy, but only focused on the currentcy conserving properties to reduce residual battery energy. Even though the currentcy conserving allocation can effectively maintain a steady currentcy allocation during the lifetime and reduce the residual battery energy at the end, one problem that can be noticed is that a task with a larger $share$ can consume less currentcy in practice, and its currentcy consumption may decrease with the increased currentcy allocation. This is a case of the priority inversion problem.

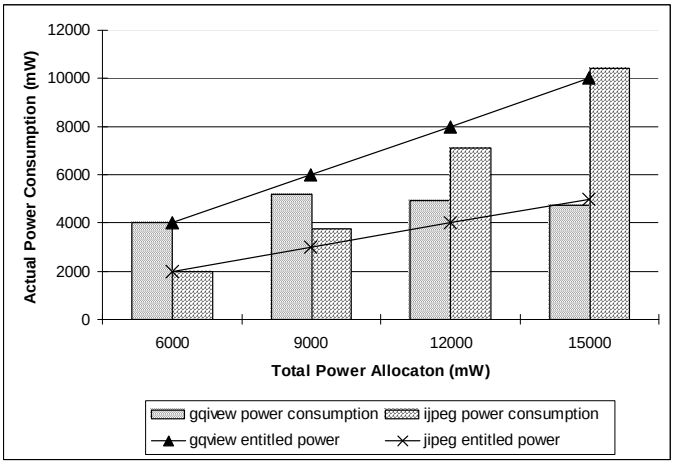
7.1.1 Priority Inversion

In general priority inversion means that a high priority task fails to run while a low priority task is executing. Priority inversion is most often caused by resource sharing. An example of priority inversion is that although a task A is given high priority for scheduling, it can be blocked by a lock that is currently held by a low priority task C. In this example, while A has high priority on the CPU, it can not preempt B on the resource of lock. The priority inversion happens if there exists another medium priority task B that does not need the lock. Task B can preempt C because of its higher priority, which will further block the highest priority task A and leaves more CPU time for task B.

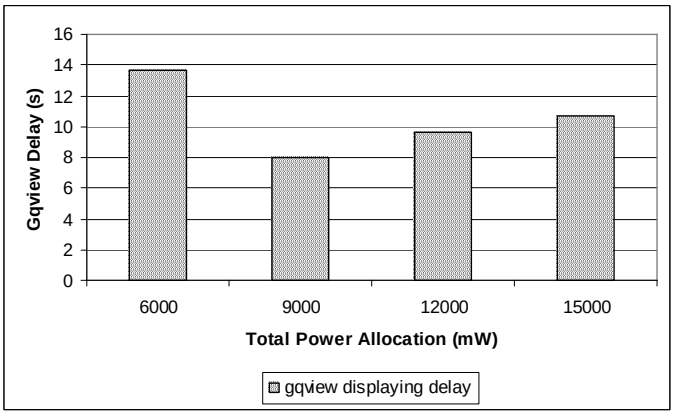
To avoid the priority inversion problem, the high priority task should be given high priority on every step on its critical path. In the previous example, task A's execution requires the lock, so the release of the lock should be given high priority. This can be realized by "giving" A's priority to task C temporarily so that it will not be preempted by task B. This method is call *priority inheritance*.

7.1.2 Allocation and Scheduling in ECOSystem

Currentcy is not an independent resource, it can only be consumed when devices are activated. Providing differentiated service is done in ECOSystem in two steps: allocation and scheduling. A large allocation of currentcy only enables the critical task to spend more, the task needs to be scheduled to run in order to consume the allocated currentcy. Compared to the priority inversion example in the previous subsection, the problem witnessed in the previous chapter is also caused by different priorities at the two steps. The allocation policy uses $share$ as the abstraction for resource sharing. With the $pay - as - you - go(PAYG)$ policy, however, ECOSystem uses the Linux scheduler which by default assumes an equal priority for all tasks.



a) Power Consumption of gqview and ijpeg



b) Gqview Delay

Figure 7.1: Currentcy Sharing with PAYG Policy

An experiment that demonstrates the priority inversion problem is shown in figure 7.1. Two tasks, gqview and ijpeg, run simultaneously with various total power allotment. The system uses the currentcy conserving allocation and $PAYG$ policy for device scheduling. In figure 7.1-a, the x axis is the total power allotment and the columns in the figure show the actual power consumption of the two tasks. The ratio of $share$ of Gqview and ijpeg is 3:2, the entitled currentcy for each of the two tasks is also plotted in the figure with lines. Figure 7.1-b shows the image displaying delay of gqview with different total power allocation.

In this experiment, when the total power is only 6,000mW, the two tasks are both mainly constrained by the currentcy resource and their currentcy consumptions match their entitled currentcy. When the total power allocation is increased to 9,000mW, the two tasks receive more currentcy and the CPU scheduling starts to be in their critical path. Even though $gqview$ has larger currentcy allocation, it has the same priority for CPU as $ijpeg$. With the competition form $ijpeg$, $gqview$ only consumes 5,164mW with the entitled power allocation of 6,000mW, while it can consume about 6,500mW running unthrottled. By throttling the power consumption of $gqview$, $Ijpeg$ can receive the 836mW "surplus" currentcy from $gqview$, which in term enables more $ijpeg$ activities and further throttles $gqview$.

As the total power allocation continues to increase, the power consumption of $gqview$ starts to decrease while ijpeg consumes more than double of its entitled share. The impact of this priority inversion problem is demonstrated more clearly in figure 7.1-b, as the delay of $gqview$ increases from 7.9sec to 10.7sec when the total power allocation increases from 9,000mW to 15,000mW.

To avoid the priority inversion problem in ECOsystem, the same priority should be provided to both currentcy allocation and currentcy consumption scheduling. The $share$ abstraction and the $share$ based allocation policy has been introduced in the previous chapter. The $share$ based scheduling policy is studied in the next section.

7.2 Currentcy Centric Scheduling

The $share$ is a task's right for currentcy resource. For allocation, it means that the task is entitled to currentcy allotment proportional to the $share$ value. For currentcy consumption scheduling, the $share$ must first be defined in terms of scheduling when tasks are competing for an energy consuming device.

7.2.1 Guidelines for Currentcy Centric Scheduler

The challenge with currentcy consumption scheduling is that it involves multiple devices and their management should be coordinated so that the total currentcy consumption could be consistent with the allocation policy.

First of all, some guidelines for schedulers is listed below that outline the opposite of *priority inversion* in ECOSystem. Here the term $giver$ is defined to be the task that gives out surplus currentcy and $receiver$ to be the task that receives surplus currentcy during the allocation. A scheduler should be able to guarantee that:

1. When two tasks, a $giver$ and $receiver$, run concurrently, the $giver$ should be given higher scheduling priority.

2. When two $givers$ or two $receivers$ run concurrently, the task with the lower currentcy consumption / share ratio should be given higher scheduling priority.

3. As long as guideline 1 and 2 are satisfied, tasks should be scheduled to consume as much of the allocated currentcy as possible.

The guidelines above are similar in form to the properties for currentcy conserving allocation in the previous chapter. Guideline 1 targets the priority inversion problem by associating scheduling priority with currentcy allocation. The idea is that once a task has received surplus currentcy, the task can not use it against the task who has supplied the currentcy. This guideline also suggests that the scheduling priority should be dynamic. Even a task with a small $share$ value could get a high scheduling priority if it supplies surplus currentcy to other tasks.

Guideline 2 states the goal of proportional currentcy consumption which is consistent with proportional currentcy allocation. The $share$ value of a task not only means the task can receive currentcy allocation proportionally, but also means it should be scheduled

to consume the resource proportionally. The scheduler that schedules tasks around the currentcy consumption is called currentcy centric scheduler.

Providing differentiated service makes sense only when the resource is competed for by tasks. Guideline 3 of resource efficiency states that idled resource should always be utilized as long as task can pay currentcy for it.

7.2.2 Architecture for Currentcy Centric Scheduling

In ECOSystem, all device schedulers should conform to the three guidelines. In this subsection a scheduling architecture is proposed that consists of setting a global scheduling priority for each task and a compliant scheduler for each device. This section focuses on the architecture and the global scheduling priority. Two device schedulers, a currentcy centric CPU scheduler and a network interface scheduler, will be introduced in the next section.

Each device scheduler only has a local view of the currentcy consumption, while the goal of currentcy consumption scheduling is the global proportionality. In the proposed scheduling architecture, device schedulers are coordinated by the global scheduling priority. A task with higher global priority means it can get more currentcy consumption on all devices. A lower global priority task gets less currentcy consumption on the shared device, but can still consume currentcy freely on the device not needed by other tasks.

When assigning the global scheduling priority, guideline 1 and 2 can be treated uniformly. ECOSystem can simply look at the ratio between a task's consumed currentcy and the entitled currentcy: $\beta_i = \frac{CurConsumed_i}{CurEntitled_i}$ and give higher scheduling priority to the task with smaller β. This method is already suggested in the guideline 2. According to the guideline 1, because $giver$ always has β smaller than 1 and β for $receiver$ larger than 1, the task with smaller β should also be given higher scheduling priority.

ECOSystem assigns the scheduling priority to tasks adaptively. It monitors the β value

of every task and increases the scheduling priority for the low β task until its β value catches up with other tasks or until the its currentcy consumption stops to respond to the increase of scheduling priority, which means that the task can not consume any more currentcy because of its own running characteristics.

7.3 Currentcy Centric Device Schedulers

This section introduces the two currentcy centric device schedulers implemented in ECOSystem: CPU scheduler and network interface scheduler. For the CPU scheduler, another feature is also introduced here, the self-pacing for low variance response time.

7.3.1 CPU Scheduler

The proposed currentcy centric CPU scheduler is based on the stride scheduler, which is a proportional, CPU-only scheduler. The stride scheduler is briefly introduced below.

Currentcy Centric Scheduling

With the stride scheduler [55], every task has a fixed priority, the number of $tickets$. Stride scheduler divides the CPU time into quanta of equal length, and they are given to ready tasks proportional to their number of $tickets$. Stride scheduler provides the service in time guarantee that if a task has one tenth of the total tickets, it then can receive one quantum every 10 time quanta. For instance if task A has 10 tickets and B has 20 tickets, the two tasks will be scheduled as the sequence (*B B A B B A B B A*...) instead of (*B B B B A A BB B B A A*...) even though they have the same ratio of 1:2. The service time guarantee is helpful in reducing response time variation caused by scheduling.

Stride scheduler provides a proportional service rate to ready tasks, but the actual CPU time may not be proportional. This is because a task has fixed scheduling priority and when it is blocked by another resource, the relinquished CPU quanta will not be compensated.

The proposed currentcy centric CPU scheduler has two major modifications to the stride scheduler. The first one is using CPU currentcy consumption as the metric to schedule processes instead of CPU time consumption. The process to be selected by currentcy centric scheduler for the next quantum is the one whose resource container has the lowest amount of currentcy spent on CPU relative to its specified $share$.

Another modification is the use of the global dynamic priority. Because the priority is based on the global currentcy consumption, if a task is blocked by another low power device, it will be compensated by the increased global priority. This approach resembles compensation tickets from lottery scheduling and fractional quanta from stride scheduling [55], both of which give an advantage toward earlier scheduling of the next quantum to a task that voluntarily relinquished part of its last quantum. Our approach is an adaptation that differs in two respects: it is based on a task's system-wide energy consumption and it applies over a longer period (spanning multiple quanta occurring during the current and previous epochs).

Scheduling for low variance response time

Given a case in which power consumption must be constrained, the epoch-based allocation has the potential to produce bursty behavior if tasks consume currentcy as quickly as they can at the beginning of an epoch and then go idle after consuming their budget. One approach to smoothing consumption rates (and as a side-effect,response times), is to shorten the epoch.

Another approach to manage the energy consumption rate is called "self-pacing" that can be incorporated into the currentcy centric CPU scheduler. The idea is to delay a task if its consumption of currentcy is ahead of schedule during an epoch. Progress is defined as the amount of currentcy spent thus far in the current epoch divided by the task's budget for the epoch. If this progress is greater than the ratio of elapsed time in this epoch over

epoch length, then the task is delayed and the processor may go idle for a short interval of time. This approach exploits the ability of currentcy to reflect an application's rate of progress. The self-pacing feature can be turned on and off independently in the currentcy centric CPU scheduler.

7.3.2 Network Interface Scheduler

In the Linux version ECOSystem is based on, there is no explicit scheduler for network interface access. By default when there are multiple TCP/IP connections, the bandwidth is equally shared among connections because of the flow control mechanism of TCP/IP, which means that the task with the most connections gets the most bandwidth. The proposed currentcy centric network interface scheduler schedules the accesses with the entity of task in stead of connection. Similar to the currentcy centric CPU scheduler, the network interface scheduler uses the global priority and schedules packets based on the task's currentcy consumption on the network interface.

One interesting challenge is the scheduling for incoming packets. The tricky issue for incoming traffic is that by the time a packet has been received and management actions can be applied, the energy to receive it has already been consumed in the wireless card. This makes it difficult to selectively receive packets destined for tasks with available currentcy as opposed to tasks without currentcy.

ECOSystem has modified the Linux network packet processing code to implement a work conserving proportional bandwidth allocation policy. This scheme identifies flows whose associated tasks have consumed bandwidth beyond their currentcy-determined share and reduces their allocated bandwidth. Assuming other tasks can consume released bandwidth, this bandwidth reduction continues until all connections consume bandwidth in proportion to their task's energy share. This is accomplished by explicitly reducing the advertised window to reduce a task's available bandwidth.

7.4 Scheduling Experiments and Results

To evaluate currency centric scheduling, this section first evaluates only the currency centric CPU scheduler and see how it is different from other performance-oriented schedulers. Then the currentcy centric CPU scheduler and the network interface scheduler are put together to demonstrate how they cooperate and how they interact with the allocation policy to achieve differentiated and yet efficient currentcy consumption. The currentcy centric CPU scheduler also incorporates the $self - pacing$ feature to smooth the throttled execution of a task, which is evaluated in the last subsection.

7.4.1 Currency Centric CPU Scheduler

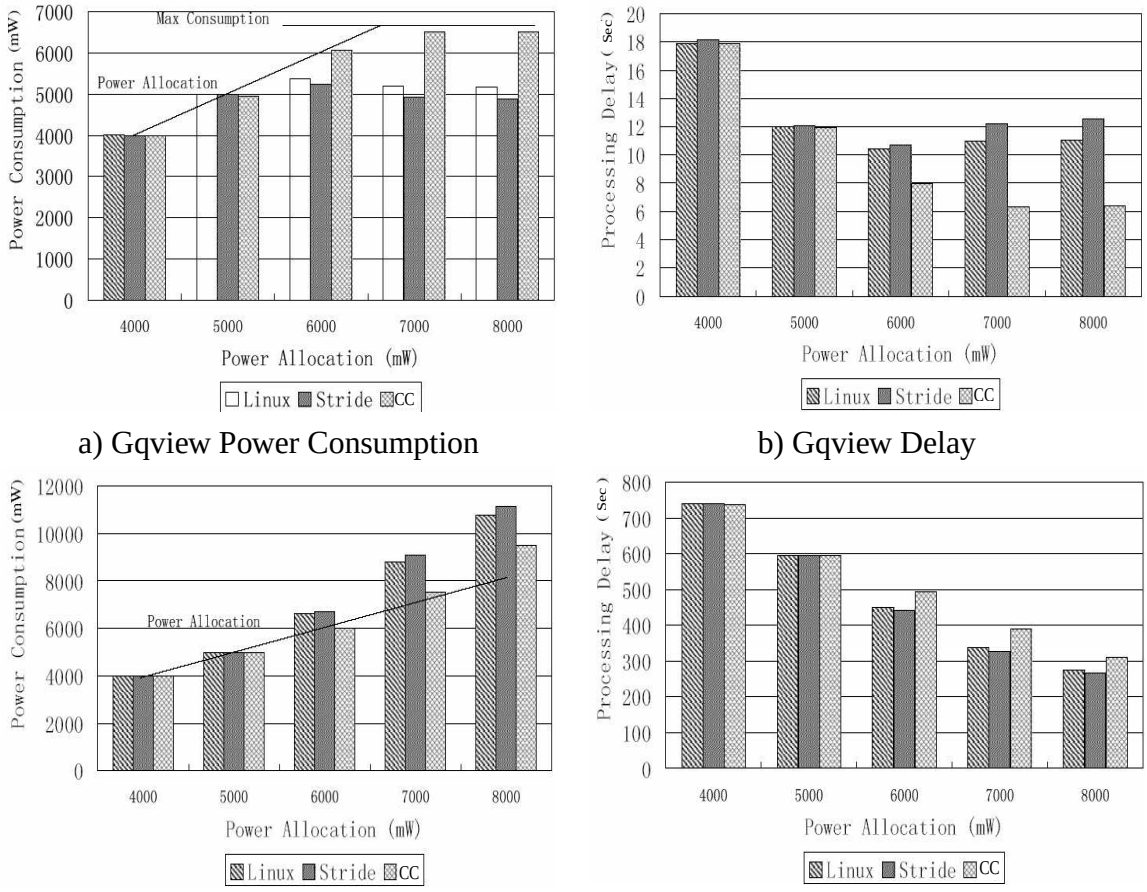
To evaluate the currentcy centric scheduler for the goal of proportional currentcy consumption, it is compared to the baseline scheduler of the $Pay - As - You - Go$ approach. With PAYG scheduling, ready processes are scheduled in a round-robin fashion until their currency runs out. According to the discussion above, the currentcy centric scheduler is unique in three important ways: 1) tasks have differentiated right for resource represented by the $share$ value, 2) currentcy consumption of the task activities is used as the metric for scheduling, 3) scheduling priorities of tasks are adjusted dynamically to achieve the definition of proportional currentcy consumption. All these three features are essential for the goal. To demonstrate the indispensability of all the three features, the currentcy centric scheduler is also compared to a currentcy-based stride scheduler. This intermediate scheduler has the first two features, but lacks the dynamic scheduling priority; It uses the $share$ value as the static scheduling priority for each task.

For the experiment presented, we simultaneously run gqview and ijpeg with equal shares of a varying total allocation. First, we execute each application alone across the range of total allocation levels to see how the performance metric associated with that

benchmark behaves. For gqview, configured with a think time of 10 seconds, the delay to completely display the given image decreases with increasing allocations of currentcy (mapped into average power for presentation) until around 6,500mW where it levels out at approximately 6.3s. We pit gqview against ijpeg, our CPU-bound benchmark that is always ready to run and whose performance metric, the delay to compress an image file, continues to decline until the maximum power consumption of the processor is reached (e.g., 15.55W). By setting the shares to be equal for the two competing applications, we are giving some benefit to the round robin and stride schedulers. In addition, the power needed by gqview for the disk (i.e., approximately 700mW) represents a relatively small level of consumption diverted to another device, making it more challenging for our currentcy centric scheduler to distinguish itself.

Figure 7.2 shows our results. Figures 7.2a and 7.2c give the power consumed by gqview and ijpeg as the allocation increases for each of the three scheduling policies. There are two additional lines on the plot for gqview showing the proportional allocation and the maximum power based on performance. Note that the bars representing the currentcy centric scheduler show that gqview receives its appropriate energy share up until the point where it approaches its maximum power requirement. The Linux default scheduler and the energy-based stride scheduler both favor ijpeg at the expense of gqview. In the case of the default Linux scheduler, this is because ijpeg is always competing with gqview for the CPU and its round-robin algorithm gives each 50% and, for gqview, that is only when it is active (not during its think time or disk access). The static currentcy-based stride scheduler experiences similar problems when gqview and ijpeg are competing for the CPU (with equal share values). Gqview is unnecessarily penalized for voluntarily reducing its energy consumption during idle periods.

From Figures 7.2b and 7.2d, we see that with the currentcy centric scheduler, gqview's delay approaches its performance of 6.3s when running without competition once it is



a) Gqview Power Consumption b) Gqview Delay

c) Ijpeg Power Consumption d) Ijpeg Delay

Figure 7.2: CPU Scheduling and Proportional Sharing of Energy

given enough power. Neither the Linux scheduler nor the stride scheduler deliver gqview that level of performance. Meanwhile, the performance level of ijpeg is appropriate to its allocation level. It benefits from redistributed currentcy once gqview's consumption levels out, exceeding its expected performance (of running alone at that allocation level) for each scheduling choice.

7.4.2 Coordinated Scheduling of Multiple Devices: Network Interface and CPU

In this experiment, to create a stressful condition for evaluation where the network is the bottleneck, we set the wireless ethernet card to 1Mbps. We execute RealPlayer, Netscape,

Application	Allocation (mW)	CPU (mW) Power	NIC (mW) Power	HD (mW) Power	Total (mW) Power	Network (B/S) Bandwidth
Energy-Centric CPU Scheduler, Energy Oblivious Network						
RealPlayer	9000	2875	219	259	3354	3066
Netscape	3000	956	569	615	2142	7294
ijpeg	3000	8960	23	0	8983	0
Default Linux CPU Scheduler, Energy-Centric Network						
RealPlayer	9000	5902	700	680	7282	8032
Netscape	3000	841	113	229	1182	2701
ijpeg	3000	6611	18	0	6629	0
Energy-Centric CPU Scheduler, Energy-Centric Network						
RealPlayer	9000	8695	621	704	10020	8353
Netscape	3000	789	155	226	1170	2680
ijpeg	3000	3778	10	0	3788	0

Table 7.1: Proportional Sharing: CPU and Network

and ijpeg with shares of 9000:3000:3000. Ijpeg serves as a CPU intensive application that does not compete for bandwidth but is capable of consuming 100% of the CPU. Realplayer plays a video clip rated at 550Kbs. Netscape continuously reloads a web page with five images with zero think time. When executing without energy constraints, RealPlayer consumes about 10,643mW to execute without pauses in video playback, while Netscape consumes 3,115mW, running unconstrained. Both RealPlayer and Netscape can consume all of the network bandwidth available.

In all of our network experiments we use the currentcy conserving energy allocation policy and constrain the total power consumption to 15,000mW. Since network bandwidth is the bottleneck resource for RealPlayer and Netscape and none of the applications' energy needs are satisfied, the goal is to achieve proportional overall energy use after satisfying the constraints of proportional network bandwidth and network energy consumption. Table 7.1 presents results for three of the scheduler design points: 1) Our energy-centric CPU scheduler with the default TCP implementation, 2) the default Linux CPU scheduler with an energy-centric network scheduler, and 3) our combined energy-centric CPU and network schedulers. We omit the case where neither the CPU or network scheduler are

energy-aware.

Our results show that the conventional network scheduler fails to provide either proportional network bandwidth or energy consumption. This is because Netscape is allowed to consume an unfair portion of network bandwidth. Netscape is able to take more than 50% of the bandwidth because it can open multiple connections. This reduces RealPlayer's ability to execute and produces an excess of currentcy that is reallocated to ijpeg. This results in ijpeg getting more of the CPU and consuming a much larger energy share than its intended allocation while the needs of the other applications are not satisfied (currentcy redistributed to ijpeg would be considered acceptable if the other applications had their needs appropriately met).

When we use an energy-centric network scheduler, but the default Linux CPU scheduler, we see that bandwidth and network power are consumed by RealPlayer and Netscape closer to the specified ratio of 3 to 1. However, RealPlayer still suffers from competition for the CPU with ijpeg which results in ijpeg significantly exceeding its energy share.

The most satisfying results are obtained by using energy-centric schedulers for both the CPU and network. Both network bandwidth and network energy are consumed proportionally by RealPlayer and Netscape. RealPlayer's share of the CPU is also protected from ijpeg by the energy-centric CPU scheduler. Netscape is throttled after receiving its share of network bandwidth (its bottleneck device) and can not consume the rest of its currentcy allocation. In this case, RealPlayer gets enough currentcy to meet its needs and execute without pausing.

7.4.3 Low Variance in Response Time Through self-pacing

In this experiment we evaluate our self-pacing scheduling and compare it to the epoch based approach. We first look at the overhead of the epoch based approach because it increases with the shortened epoch length and could become a performance bottleneck.

Scheduling	Power Consumption (mW)				Delay (seconds)			
	CPU	Disk	Network	Total	Min	Max	Average	Std. Dev
Unthrottled	1047	1013	136	2197	0.27	0.68	0.43	0.11
Epoch	351	812	48	1212	1.0	33.8	3. 8	5.8
Self-Pacing	313	842	43	1199	3.3	5.6	4.0	0.6

Table 7.2: Netscape Response Time Variation

However, experiments show the overhead for currentcy allocation is very small. Even if we perform allocation every 10ms (a timer interrupt occurs every 10ms, while the CPU scheduling quantum is 60ms in our system), the overhead is only 206μs for 18 resource containers and 40μs for 3 containers.

To explore the effects on response time of our two approaches for reducing bursty performance, we run Netscape and continuously load our department's web page. This page contains a banner image and some simple text. The autoload is implemented using a javascript and this also allows us to measure the page load latency. This latency is composed of several http requests, displaying the content and updating the disk cache. We set the think time between successive page loads at 2 seconds. Executing without any throttling requires about 2,197mW.

We evaluate both an epoch-based approach that uses 0.01 second epochs, and the self-pacing approach with 10 second epochs. We allocate currentcy equivalent to an average of 1,200mW to Netscape and measure 54 consecutive page loads for the self-paced test and 41 page loads for the epoch based approach. Differences are apparent in Table 7.2 when we examine the delay for a page load. Although the average delay is similar for the two policies, the self-paced scheduler has much lower variation in the delay. This can translate into a user perceived difference in performance as the self-pacing policy can provide a visibly smoother display of the web page. We note that similar visible differences occur when executing other applications, such as RealPlayer, Acrobat, and StarOffice.

7.5 Summary

Even though currentcy is allocated to tasks proportional to their $share$ value, the goal of providing differentiated service to tasks can not be achieved without scheduler support. In one experiment, it is shown that a small $share$ task can use its scheduling priority to "steal" currentcy from another large $share$ task. This problem is identified as a case of the priority inversion problem.

The priority inversion problem is caused by the currentcy-oblivious scheduler that is inconsistent with the allocation policy. In this chapter, a currentcy centric scheduler is proposed. The basic idea is that if a task gives out surplus currentcy, then it should be protected from the tasks that receive its currentcy. In the currentcy centric scheduler, each task has a global priority for its scheduling on every device. If a task gives out surplus currentcy, its global scheduling priority will be increased to protect this task and if a task receives surplus currentcy, its global scheduling priority will be decreased to prevent it from violating other tasks.

Two device schedulers, a CPU scheduler and a network scheduler, are implemented that use the global priority for their scheduling. The experiment results show that with currentcy centric scheduling, the priority inversion problem identified here can be eliminated. It is also shown that an application's demand for multiple resources can be protected with our unified scheduling effort.

Chapter 8

Interaction with Hardware and Applications

Our work on energy management has been focusing on the framework and policies to achieve global energy-related goals without requiring applications to be modified or energy-aware. To enforce the global goals, the currentcy model is used to control the energy consuming activities of every application and device. In this chapter, we are interested in using currentcy as the model to communicate among applications and devices for making better decisions. The basic idea is that applications and devices can be given more flexibility within the framework as long as the global goals are not violated. In this chapter we will first introduce how currentcy information can be used in a hard disk access scheduler to reduce the energy cost per access. Then we will discuss how an application can interact with the OS for more efficient currentcy utilization.

8.1 Disk Access scheduling

Encouraging more energy efficient use of devices is an important function of an energy centric operating system. Currentcy provides a means for passing along the savings to tasks that cooperate through their usage patterns. The disk presents unique challenges and opportunities for energy efficiency since it has non-uniform power consumption. The cost of spinning up the disk is much greater than keeping it spinning for a short duration. In this section, we consider techniques for more efficient disk access, focusing on sharing the spinup/spindown power costs.

The policy space for energy efficient hard disk access scheduling is very large, and many solutions may require an API for application involvement. For example, recent work [58] describes cooperative disk I/O operations that applications can use to facilitate

such behavior. In this paper, we have limited our studies to techniques for managing disk access using currentcy model that can be implemented solely within the operating system without application involvement.

8.1.1 Shaping Access Patterns by Pricing and Bidding

Intuitively, we want to amortize spinups across multiple disk operations, which benefits from encouraging more bursty behavior. The key to more effectively manipulating the spinup/spindown behavior is *shaping the disk access patterns* to take advantage of this cost-sharing benefit within the debiting policy. Here we present a set of pseudo mechanisms that shape the access patterns through debiting, bidding and pricing in the context of our currentcy model.

Pricing disk accesses can be used to reward a task for performing disk accesses in bursts. One approach we investigate sets the *entry price* of a disk access that requires a spinup cost much higher than the actual cost. When the access is actually permitted, we then debit the actual cost. This forces the task to accumulate enough currentcy to ensure that it can execute for a reasonable amount of time following the first access in hopes of generating more disk accesses while the disk is spinning.

We augment this pricing policy with the ability of tasks to bid on disk accesses. Tasks can indicate they are willing to contribute certain amounts toward the price of spinning up the disk. This is a natural place for API extensions. However, the OS can apply this technique transparently by checking the task's budget for sufficient surplus, analogous to a credit check. One goal of this technique is to enable multiple tasks to pool their currentcy and cooperatively use the disk in an energy-efficient manner.

Traditional techniques of skewing access patterns are amenable to currentcy-based variations. These include exploiting block caching and delaying writes while the disk is not spinning, piggybacking prefetching upon requests that spin up the disk on demand, and

managing the buffer allocation. Thus, we explore a buffer allocation policy tied to the average disk access cost. Subject to limitations on the number of buffers systemwide, this policy attempts to reduce the costs (via effective prefetching and delayed writes) and make them uniform across tasks (which can tend to synchronize tasks into producing batches of disk activity).

We trigger prefetching operations and flushing of delayed writes that cause spinups using a bidding function based on the fraction of consumed buffers. The range of potentially useful bidding functions is not investigated in this thesis. We provide results for one bidding function that sets a bid offer to zero if less than 80% of the prefetch buffers are consumed otherwise to a weighted linear value ($bid = entry\_price * (percent\_buffers\_consumed - 80)/(100 - 80)$). This corresponds to a function where value is greatly increased as the task nears a demand fetch. The disk flush daemon performs a large number of writes once it starts flushing pages to a spinning disk, writing back all dirty pages that have been idle for a more than 5 seconds. By contrast, the default Linux page flush policy is to check every 5 seconds for dirty pages that have not been accessed for 30 seconds and write those to disk.

8.1.2 Experiments on Disk Access Scheduling and Buffer Allocation

In this section, we show how the currentcy model enables policies based on pricing and bidding. First, we explore techniques to coschedule disk accesses for two applications with the goal of reducing overall disk power consumption. We present preliminary results on prefetching in our coordinated buffer management system.

Accesses In our first experiment we execute ijpeg and gqview concurrently, and each application demand fetches data from the disk. Ijpeg performs image compression on a set of ppm format image files. Each file is a copy of the same SPEC input (command line

options: -GO.compress vigo.ppm) that is 2,359,355 bytes. When running unconstrained, ijpeg requires about 2.452 seconds to process each file and start to read the next. Gqview displays the same set of image files using autobrowse mode with a 50 second think time. We set the total power allocation to 1,500mW and the two tasks each get an equal share of 750mW. These severe constraints are used to accentuate the disk's impact on performance. The entry price for initiating a disk access is set to 24,000mJ (twice the combined cost of spinup and spindown). We use an immediate disk spindown.

Without pricing/bidding, the total average disk power consumption is 911mW, with 403mW and 508mW for ijpeg and gqview, respectively. Our currentcy-based policy formulated in terms of pricing/bidding reduces this value to 655mW (313mW ijpeg and 342mW gqview) by engineering more task cooperation in disk spinup sessions. Furthermore, the performance of each application improves, particularly ijpeg which requires only 57 seconds to process the file compared to 74 seconds without pricing/bidding.

The next experiment is designed to show the benefits of bidding for energy efficient disk prefetching. We set the total allocation at 1,500mW and execute ijpeg (same input as above) with 300mW concurrently with the MP3 player, x11amp, which receives 1,200mW allocation. X11amp reads a 3MB file, and is amenable to prefetching because of its sequential access pattern. We use the combined pricing/bidding approach where there is a high entry price for a disk spinup and x11amp contributes by bidding based on its prefetch buffer consumption. The average total disk power consumption is 357mW compared to 565mW without pricing/bidding. Ijpeg's average disk power consumption reduces from 365mW to 229mW and its performance improves from 90 seconds per file to 66 seconds. X11amp's disk power consumption reduces from 200mW to 128mW, and it does not incur any pauses in either policy.

Buffer Allocation It is also effective to balance the buffer allocation among prefetching-friendly tasks to facilitate more globally synchronized disk activity. To show the benefit of cooperation, we compare local and global prefetching behavior for two applications (x11amp and StarOffice) that exhibit sequential access patterns, since the unified buffer cache in Linux can easily detect these sequences and initiate prefetching. The spindown timeout is set to 1 second. X11amp reads a 3MB file, while StarOffice reads an 11MB presentation with 14 slides and executes in auto-transition mode with approximately 20 seconds between slides.

When prefetching is performed locally using the default Linux buffer allocation of 32 buffers for each task, the spinup and spindown costs are incurred for each task. The disk power consumption for x11amp is 186mW and StarOffice consumes 219mW for a combined 405mW.

In contrast, a global prefetching policy synchronizes the prefetching operations of the two tasks by allocating prefetch buffers according to a task's average disk access cost, which is determined by the task's buffer consumption rate. In this experiment, x11amp requires 256 prefetch buffers and StarOffice uses 1000. This significantly reduces the total disk power consumption to 280mW, with 65mW for X11amp and 215mW for StarOffice. StarOffice receives very little benefit since it is dominated by the cost to actually read the data, whereas X11amp leverages StarOffice's relatively large number of disk accesses.

8.2 Application-OS Interaction within the Management Framework

Previously we have assumed no application involvement in the currentcy management because modifying applications to be currentcy-aware incurs large cost and we'd like to see what can be done by the OS alone. We have shown that some application information may be estimated by the OS and adaptive approaches can be used for dynamic application behavior. However some valuable information, such as the application performance and how

it correlates with its energy constraint, is very difficult to be inferred by the OS. Without such information, energy is managed for the sake of energy goals, totally disregarding the impact on performance.

At the application side, without OS interface, resource constraint is not exposed to applications for better adaptation. Also an application has no way of expressing its unique needs to the OS, so every application is treated with the same management policy. An application and the OS can interact in many different ways and at different levels. In this section we discuss two examples of such interaction and show how the OS can provide better service with some simple application cooperation.

8.2.1 Currentcy-based Admission Control

When sharing currentcy, our energy management framework emphasizes the protection of the entitled resource for each task. The entitled resource should not be violated as long as it can be consumed by the task. However there are some applications that, on certain conditions, may relinquish its entitled currentcy voluntarily even though it is still able to consume the currentcy. For example $x11amp$ requires 80mW to decode and play an MP3 audio stream on our hardware platform. When its power allocation is below the threshold value, its execution will be throttled and silent pauses during the playback will result, which is intolerable for most listeners. $X11amp$ can inform this minimal currentcy requirement to the OS that when it can not be met, the currentcy can be given to other applications that do need the resource.

ECOsystem can add a layer of currentcy based admission control in its currentcy management. The admission control has three components: the interface, the admission policy and the callback mechanism. When an application is started it should first specify its minimal currentcy requirement to the OS through the interface and request for admission. The request is either granted or rejected by the admission control policy based on whether or

Task	Min Power	Entitled Power	Power Allocation W/O Admission Control	Power Allocation W Admission Control
X11amp	80mW	50mW	50mW	80mW
Netscape	220mW	60mW	60mW	0mW
Ijpeg	0mW	50mW	50mW	80mW

Table 8.1: Currentcy Based Admission Control Example

not its minimal requirement can be met. The admission policy has to re-evaluate the decision for every task once there is a new task started or old task terminated. When there is a change in the admission status, the callback mechanism can be used to notify the application of such change. Note that our currentcy based admission control is still consistent with the management framework, only with the added ability for a task to specify the condition on which it gives up the entitled currentcy for the benefit of other tasks.

Table 8.1 shows an example of the admission control policy. The first column lists the three applications in this example, $x11amp$, $netscape$ and $ijpeg$. The second column shows the minimal power requirement for the applications. The minimal power for $netscape$ is set to be 220mW, corresponding to a 13.62 sec delay. Any delay longer than 13.62 sec is considered intolerable for the user. The last application, $ijpeg$, has no minimal power requirement. The third column is the entitled power for the applications based on their resource $share$. We can see that both $x11amp$ and $netscape$ have entitled power allocations below their minimal requirements. Without currentcy based admission control, every application simply runs with the allocated power, which means intolerable performance for both $x11amp$ and $Netscape$. With currentcy based admission control, $Netscape$ can be rejected admission (based on the criteria of $currentcy_{minimal}/currentcy_{entitled}$ for example). With $netscape's$ currentcy going to other applications, x11amp is granted admission and can run without silent pause.

In addition to minimal resource requirements, some applications have discrete per-

formance levels and thus discrete resource requirements. For these applications it is not necessary to allocate resource in between these levels. For example $Realplayer$ may have choices of several fidelity levels for the same content. The OS can allocate currentcy to meet the highest fidelity level within its share and allocate the extra currentcy to other tasks.

8.2.2 Application Involvement in the Energy Management

Our currentcy based admission control shows an example of more efficient energy management based on one piece of information (minimal resource requirement) from the application. Within our energy management framework, a currentcy-aware application can further interact with energy management policies for its unique need. For instance without application participation, there is only one policy in ECOSystem to manage the limited currentcy resource for each application: guaranteeing the execution time by throttling the application's energy consumption rate. While this default policy works for some applications, such as $ijpeg$ and a background spell checker, some applications are better served with other policies such as the guarantee for a certain energy consumption rate with shortened execution time. For example, instead of running throttled with 40mW power allocation for the target lifetime of 2 hours, $x11amp$ can request to run with 80mW (the minimal resource requirement) for 1 hour. For $gqview$, anticipating smaller image files to be displayed at the end of the browsing, it may request larger power allocation for the first hour and smaller power allocation for the second hour. All these requests are still within our energy management framework, but application's performance will be improved with policies tailored for each application.

Upon receiving a request from the application, the OS may need to consider the risk associated with granting the request. For example if the OS allows currentcy to be over-drawn by $gqview$ for the first hour, it may not have enough currentcy to serve other unexpected

applications or events later. This situation is similar to a bank loaning out some money at the risk of not being payed back. The OS can take into account the application's income (currentcy allocation every epoch) in its decision, or it can impose an interest rate for the loan to establish a fund for unexpected events.

8.3 Summary

The energy resource is a critical resource to the performance of hardware devices and applications. Thus the currentcy information can be used as a prediction for the activities of devices and applications in the system. In this chapter, a hard disk access scheduler is proposed that utilizes the currentcy information to increase the hard disk access energy efficiency. The hard disk access cost can be reduced by shaping the hard disk access pattern of a task to be more energy-friendly, or by grouping accesses from different tasks to share the access cost. Experiment results show that with a set of currentcy-based pseudo-economic mechanisms, the disk access cost can be greatly reduced for both synchronous and asynchronous disk accesses.

Currentcy is a powerful model that can be used to formulate various policies. The hard disk access scheduler is an example of such policy within the management framework. The possibility of using the currentcy model for OS-application interaction is also discussed in this chapter. It is shown that with some application involvement, many policies can be formulated to meet applications' special demands.

Chapter 9

Related Work

This chapter discusses a variety of research related to the work of this thesis. The related work is classified into several topics, including the energy management at the system level, the energy management at the device level, the industrial efforts on energy management, device-specific energy-saving efforts and resource sharing and scheduling.

9.1 System Level Energy Management

Work by Flinn and Satyanarayanan on energy-aware adaptation using Odyssey [21] is closely related to ECOSystem in several ways. Their fundamental technique differs in that it relies on the cooperation of applications to change the fidelity of data objects accessed in response to changes in resource availability. The goal of one of their experiments is to demonstrate that by monitoring energy supply and demand to trigger such adaptations, their system can meet specified battery lifetime goals before depleting a fixed capacity and without having too much residual capacity at the end of the desired time (which would indicate an overly conservative strategy). They achieve a 39% extension in lifetime with less than 1.2% of initial capacity remaining. For their approach, the performance tradeoff takes the form of degraded quality of data objects. In contrast, the work of this thesis focuses on managing global system resources in a unified manner. Unmodified applications and those that are not necessarily able to change "fidelity" benefit from our approach. Overall, the two efforts can be viewed as complementary: the operating system should manage global system devices in response to application requirements and the application should adapt its behavior when appropriate to reduce energy consumption.

Also closely related to this effort is work by Ma and Shin [36]. They conduct a simu-

lation study based on the workload presented by Ellis [20] to evaluate a new energy-aware scheduling discipline based on the Emeralds [61] OS framework. Their fundamental technique is to stretch the period of low-priority tasks to extend the battery lifetime. Thus, their scheduler considers not only the priority of a task but also its total energy consumption in determining how often the task runs as a function of remaining battery power. As the battery drains, energy considerations are given increasing weight. Relative to their approach, this thesis built a working operating system to experimentally measure the impact of various allocation and scheduling policies on energy. Further, extending battery lifetime is achieved through limiting the maximum drain rate of the battery by explicit energy management.

Explicit energy management has also been designed for the Nemesis operating system [44]. This proposal describes how to extend the Nemesis resource accounting mechanisms, based on a calibration of device power consumption, to account for energy use by applications. Resource management is similar to Odyssey in that it is based on collaboration with applications. In Nemesis, this takes the form of an economic model for providing feedback to processes that allow them to adapt to shortages in energy availability.

9.2 Industrial Standard

Attention to the issues of energy and power management is gaining momentum within both academics and industry. APM(Advance Power Management BIOS Interface) and ACPI(Advanced Configuration and Power Interface) are two generations of a power management interface. They both emphasize the standardization of the energy management interface, with the insight that energy management is a system wide effort, even for the management of a single device or application. ECOSystem focuses on the energy management framework and management policies, and can benefit from the standardized device control defined in these two interfaces.

APM and ACPI are different in architecture. In APM, many of the power management functionalities and management policies are implemented in the BIOS, which is supplied by the OEM and specific to the hardware platform [3]. The APM BIOS manages power in the background based on device activity, and can operate independently without the OS. APM-aware applications and operating system can participates in the management via an APM driver, which calls the APM BIOS through the standard interface.

ACPI is established for OS-directed configuration and power management that integrates changing energy management technologies with a standard interface [1]. This is done by embedding AML(ACPI Machine Language) into ACPI compatible devices. Basically, AML is a small program inside the hardware implementing the function calls defined in the ACPI. The OS can simply invoke the corresponding AML code for a certain function to avoid the low-level details of the hardware. Compared to the APM, in ACPI the OS has direct control of devices and can better coordinate the activities of applications.

9.3 Device Energy Consumption Management

Energy is also a challenge for the design of many devices because of the problem of heat dissipation. There has been previous work on limiting CPU activity levels, in particular for the purpose of controlling processor temperature, via the process management policies in the operating system. In [43], the operating system monitors processor temperature and when it reaches a threshold, the scheduling policy responds to limit activity of the "hot" processes. A process is identified as "hot" if it uses the CPU extensively over a period of time. As long as the CPU temperature remains too high, these hot processes are not allowed to consume as much processor time as they would normally be entitled to use. This work only considers the power consumption of the CPU as opposed to a total system view. This strategy was implemented in Linux and results show that power constraints or temperature control can be successfully enforced. The performance impact is selectively

felt by the hot processes which are likely not to be the foreground interactive ones.

The idea of performing energy-aware scheduling using a throttling thread that would compete with the rest of the active threads has been proposed by Bellosa [7]. The goal is to lower the average power consumption to facilitate passive cooling. Based upon his method of employing event counters for monitoring energy use, a throttling thread would get activated whenever CPU activity exceeded some threshold. When the throttling thread gets scheduled to run, it would halt the CPU for an interval.

9.4 Energy Efficient Hardware Access

Most of the literature on power/energy management has been dominated by consideration of individual components, in isolation, rather than taking a system-wide approach. The term "throttling" (which is used in this thesis in a very general sense) is most often associated with the growing literature on voltage/clock scheduling [40, 41, 25, 57, 24, 59, 30, 56] for processors that support dynamic voltage scaling. Here, the "scheduling decision" for the OS is to determine the appropriate clock frequency / voltage and when changes should be performed. Interval-based scheduling policies track recent load on the system and scale up or down accordingly. Task-based algorithms associate clock/voltage settings with the characteristics (e.g. deadlines, periodic behavior) of each task.

Disk spindown policy has been studied in [18, 27, 32, 34]. Without interaction with applications, these work study the problem of choosing the best power state with a static trace of hard disk accesses. [58] describes techniques involving buffer management policies and an API allowing application cooperation for shaping the disk request pattern to increase the effectiveness of disk spindown. This body of work is complementary to our currentcy model, as illustrated by our incorporation of spindown policies, and will impact the debiting policies for such devices in our framework.

Lowering the power state of a hard disk can save energy while the disk is idle, but

it must pay a one-time cost for the state transition. This problem can be generalized as a Rent-To-Buy decision [32] that also exists in the management of other devices such as memory page allocation [16, 33] and wireless networking protocols [31, 48]. The emphasis in much of this work has been on dynamically managing the range of power states offered by the devices.

9.5 Resource Sharing and Scheduling

ECOSystem incorporates several ideas from previous work. The idea of currentcy borrows from the tickets abstraction of lottery scheduling [54, 55] with the value of a currentcy unit tied to energy. The key insight that distinguishes ECOsystem is that energy normalizes resource management across the diverse set of devices that consume it. ECOSystem adopts the resource container abstraction [4] as the accounting entity in order to allow more complete accounting of activity and lazy receiver processing [19] in accounting for packet processing overhead.

ECOSystem uses a simple proportional share scheduler that allocates currentcy to logical tasks on a per-epoch basis. Once a task's total energy consumption exceeds its allocated currentcy (based on its CPU, disk, network, etc. activity), the ECOSystem scheduler puts all processes associated with that task to sleep. Our scheduler builds upon existing work in proportional share allocation in CPU [55, 4] and network scheduling [17].

Chapter 10

Conclusions

The first section of this chapter summarizes the work in the previous chapters. Several future research directions are opened up with the work of this thesis, which are discussed in section 2.

10.1 Summary

The utility of emerging computing environments is increasingly limited by available energy rather than raw system performance. There have been many efforts to limit the energy usage of specific hardware devices. In this thesis it is advocated that energy should be explicitly managed as a first-class system resource that cuts across all existing system resources, such as CPU, disk, memory, and the network in a unified manner. Thus, a unifying abstraction is proposed to integrate power management into the two traditional tasks of the operating system, 1) hardware abstraction and 2) resource allocation. This allows the OS to reason about the overall energy behavior of an application on a platform-specific basis and to potentially extend the useful lifetime of the system for unmodified applications.

This thesis tackles one of the most important problems haunting the design of battery-powered mobile and wireless computing systems - the management of battery lifetime and the energy/power resource, in general. As the basic model for energy management, the currentcy model is proposed that unifies diverse hardware resources under a single management framework. A prototype energy-centric operating system, ECOSystem, is implemented that incorporates the currentcy model and demonstrates techniques for explicit energy management with a total system point of view. This system is then applied toward the specific goals of achieving a target battery lifetime and prioritized energy sup-

ply among competing applications. ECOSystem also provides a testbed for formulating various resource management policies in terms of currentcy.

The embedded energy model approach is used in ECOSystem. To address the two dimensional distribution of the energy resource and the asynchronous application activities, the embedded energy model approach consists of three components, activity tracking, device energy model and payback policy. Application activities are tracked to associate device activities with the applications responsible. A device-specific energy model is used to calculate the energy consumption incurred, and the payback policy that attributes the energy consumption to the applications is identified.

Several currentcy-based policies are formulated within ECOSystem, including the currentcy conserving allocation policy that reduces residual battery capacity, the currentcy centric scheduling policy that schedules application activities around the energy resource, and the self-pacing execution to smooth response time variation. The fact that all these policies are based on currentcy demonstrates that currentcy is a powerful model for expressing energy management policies. Experiment results show that these currentcy-based policies, being able to capture aspects of global energy use, provide more coherency to system-wide energy management.

Energy-saving efforts at hardware devices, although not a part of the energy management framework, can also benefit from the currentcy model. This is demonstrated in this thesis by the energy efficient hard disk access scheduling through currentcy-based bidding, pricing and debiting. Using the currentcy information, the hard disk access scheduler can shape the disk access pattern of applications to create energy-saving opportunities for the disk spindown policy. The potential benefit of exposing the currentcy information to applications is also discussed in this thesis.

10.2 Future Directions

One interesting avenue of future research suggested by the results of this thesis is to further consider the interaction between ECOSystem and the applications. While an explicit goal of this thesis has been to understand the potential benefits of OS energy management with unmodified applications and current hardware, ultimately maximum benefits can only be achieved through OS-application cooperation.

Another direction of future research is to study the interaction between ECOSystem and the hardware platform. In fact, one immediate observation of this thesis is that reducing the baseline power consumption in future hardware platforms, will result in significant additional operating system opportunities to manage energy. Device management within ECOSystem is also an interesting area, for example a currentcy-based DVS algorithm may reduce the CPU frequency to throttle the application instead of inserting idle quanta.

Because the energy resource co-exists with other devices, their management should be considered jointly. In ECOSystem, devices are scheduled around the energy resource with the assumption of energy being the most limiting factor in the system performance. A promising research area is the development of a resource sharing mechanism that coherently handles energy and other resources. One possible approach to do this is to provide the OS interface for applications to negotiate their resource requirements within the resource constraints.

Bibliography

[1] ACPI. *Advanced Configuration and Power Interface Specification*, Oct. 2002. www.acpi.info.

[2] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 176–189. ACM Press, 2003.

[3] APM. *Advanced Power Management (APM) BIOS Interface Specification*, Feb. 1996. http://download.microsoft.com/download/whistler/hwdev3/1.0/WXP/EN-US/apmv12.exe.

[4] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Third Symposium on Operating Systems Design and Implementation*, February 1999.

[5] Kenneth C. Barr and Krste Asanovic. Energy aware lossless data compression. In *MobiSys 2003*, 2003.

[6] Smart Battery. *Smart Battery Data Specification*, DEC. 1998. http://www.sbs-forum.org.

[7] F. Bellosa. The benefits of event-driven accounting in power-sensitive systems. In *Proceedings of the SIGOPS European Workshop*, September 2000.

[8] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A discrete-time battery model for high-level power estimation. In *Proceedings of DATE*, pages 35–39, 2000.

[9] J. Blanquer, J. Bruno, E. Gabber, M. McShea, B. Ozden, A. Silberschatz, and A. Singh. Resource management for qos in eclipse/bsd. In *in Proc. of FreeBSD'99 Conf., Berkeley, CA, USA, Oct. 1999.*, 1999.

[10] T. Burd and R. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proc of 28th HICSS Conf, Vol. I*, pages 288–297, January 1995.

[11] E. Chan, K. Govil, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proc. of the 1st ACM International Conf. on Mobile Computing (MOBICOM95)*, pages 13–25, November 1995.

[12] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-Power CMOS Digital Design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[13] F. Chang, K. Farkas, and P. Ranganathan. Energy-Driven Statistical Profiling Detecting Software Hotspots. In *Workshop on Power-Aware Computer Systems*, February 2002.

[14] C. F. Chiasserini and R.R. Rao. Pulsed battery discharge in communication devices. In *Proceedings of MOBICOM*, August 1999.

[15] C. F. Chiasserini and R.R. Rao. Energy efficient battery management. In *Proceedings of INFOCOM*, March 2000.

[16] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramiam, and M.J. Irwin. DRAM energy management using software and hardware directed power mode control. In *Proceedings of 7th Int'l Symposium on High Performance Computer Architecture*, January 2001.

[17] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM 1989*, 1989.

[18] Fred Douglis, P. Krishnan, and Brian Bershad. Adaptive Disk Spin-down Policies for Mobile Computers. In *2nd USENIX Symposium on Mobile and Location-Independent Computing*, April 1995. Monterey CA.

[19] Peter Druschel and Gaurav Banga. Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems. In *Operating Systems Design and Implementation*, October 1996.

[20] C. S. Ellis. The Case for Higher-Level Power Management. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems*, Rio Rico, AZ, March 1999.

[21] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles (SOSP)*, pages 48–63, December 1999.

[22] Jason Flinn and M. Satyanarayanan. PowerScope: A tool for profiling the energy usage of mobile applications. In *Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 2–10, February 1999.

[23] S. Gold. A pspice macromodel for lithium-ion batteries. In *12th Annual Battery Conference on Applications and Advances*, pages 215–222, Jan 1997.

[24] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithm for dynamic speed-setting of a low-power cpu. In *Proceedings of first annual international conference on Mobile computing and networking*, November 1995.

[25] D. Grunwald, P. Levis, K. Farkas, C. Morrey, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)*, October 2000.

[26] S. C. Hageman. Simple pspice models let you simulate common battery types. In *the Journal of EDN*, pages 117–132, Oct 1993.

[27] D. Helmbold, D. Long, and B. Sherrod. A Dynamic Disk Spin-Down Technique for Mobile Computing. In *Proc. of the 2nd ACM International Conf. on Mobile Computing (MOBICOM96)*, pages 130–142, November 1996.

[28] IBM. *IBM Travelstar 25GS, 18GT, and 12GN 2.5-inch hard disk drives*, Jan. 2000. http://www.hgst.com/tech/techlib.nsf/products/Travelstar_12GN.

[29] Intel. *Pentium III Processor for the PGA370 Socket at 500 MHz to 1.13GHz*, June 2001. http://developer.intel.com/design/pentiumiii/datashts/245264.htm.

[30] Tohru Ishihara and Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of 1998 International Symposium on Low Power Electronics and Design*, pages 197–202, August 1998.

[31] R. Kravets and P. Krishnan. Power Management Techniques for Mobile Communication. In *Proc. of the 4th International Conf. on Mobile Computing and Networking (MOBICOM98)*, pages 157–168, October 1998.

[32] P. Krishnan, P. Long, and J. Vitter. Adaptive Disk Spin-Down via Optimal Rent-to-Buy in Probabilistic Environments. In *Proceedings of the 12th International Conference on Machine Learning*, pages 322–330, July 1995.

[33] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla S. Ellis. Power aware page allocation. In *Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS IX)*, pages 105–116, November 2000.

[34] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *USENIX Association Winter Technical Conference Proceedings*, pages 279–291, 1994.

[35] J.S. Newman M. Doyle, T.F. Fuller. modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. In *J.Electrochem. Soc. Vol.140 pp1526-1533*, 1993.

[36] Trolan Ma and Kang G. Shin. A User-Customizable Energy-Adaptive Combined Static/Dynamic Scheduler for Mobile Applications. In *Proceedings of IEEE Real-Time Systems Symposium*, November 2000.

[37] Peter Marbach. Priority service and max-min fairness. *IEEE/ACM Trans. Netw.*, 11(5):733–746, 2003.

[38] T. Martin and D. Siewiorek. Non-ideal battery properties and low power operation in wearable computing. In *Proceedings of the 3rd International Symposium on Wearable Computers*, October 1999.

[39] Debashis Panigrahi, Carla Chiasserini, Sujit Dey, Ramesh Rao, Anand Raghunathan, and Kanishka Lahiri. Battery life estimation of mobile embedded systems. In *14th International Conference on VLSI Design (VLSID 2001)*, 2001.

[40] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lpARM microprocessor system. In *Proceedings of International Symposium on Low Power Electronics and Design*, 2000.

[41] Trevor Pering, Thomas D. Burd, and Robert W. Brodersen. The Simulation and Evaluation of Dynamic Scaling Algorithms. In *Proceedings of the International Symposium on Low Power Electronics and Design*, August 1998.

[42] David Petrou and John Milford. Proportional-share scheduling: Implementation and evaluation in a widely-deployed operating system.

[43] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *Proceedings of 2nd Workshop on Feedback Directed Optimization*, November 1999.

[44] Derek McAuley Rolf Neugebauer. Energy is just another resource: Energy accounting and energy pricing in the nemesis os. In *HOTOS 2001*, May 2001.

[45] Alexey Rudendo, Peter Reiher, Gerald Popek, and Geoffrey Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review, SIGMOBILE*, 2(1):19–26, January 1998.

[46] SANYO. *Cell Type UF103450P(1700mAh) Specifications*, AUG. 2002. http://sanyo.wslogic.com/pdf/pdfs/UF103450P.pdf.

[47] S. Schlosser, J. L. Griffin, D. Nagle, and G. Ganger. Designing computer systems with MEMS-based storage. In *Proceedings of Ninth Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[48] Mark Stemm and Randy Katz. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. In *Proceedings of 3rd International Workshop on Mobile Multimedia Communications (MoMuC-3)*, September 1996.

[49] C-L Su and A. Despain. Cache Designs for Energy Efficiency. In *Proc. of 28th Hawaii International Conf. on System Science (HICSS)*, January 1995.

[50] David G. Sullivan and Margo I. Seltzer. Isolation with flexibility: A resource management framework for central servers. pages 337–350.

[51] Mani B. Srivastava Sung Park, Andreas Savvides. Battery capacity measurement and analysis using lithium coin cell battery. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 382–387. ACM Press, 2001.

[52] T13. *ANSI X3.298-1997*, 1997. http://www.t13.org.

[53] V. Tiwari, S. Malik, and A. Wolfe. Compilation Techniques for Low Energy: An Overview. In *Proc. of the 1994 IEEE Symp. on Low Power Electronics*, pages 38–39, October 1994.

[54] C. Waldspurger and W. Weihl. Lottery scheduling: Flexible proportional share resource management. In *Proceedings of Symposium on Operating Systems Design and Implementation*, November 1994.

[55] C. A. Waldspurger. Lottery and stride scheduling: Flexible proportional-share resource management. Technical Report MIT/LCS/TR-667, 1995.

[56] Klara Nahrstedt Wanghong Yuan. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *SOSP03*, Oct 2003.

[57] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of First Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994.

[58] 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.

[59] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE Symp. on Foundations of Computer Science*, October 1995.

[60] Lixia Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *ACM Transactions on Computer Systems*, May 1991.

[61] Khawar M. Zuberi, Padmanabhan Pillai, and Kang G. Shin. Emeralds: a small-memory real-time microkernel. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 277–299. ACM Press, 1999.

Biography

Heng Zeng was born in Nanchang, China, on Dec 18, 1973. He was awarded a Bachelor of Science degree in computer science and technology from Tsinghua Unive, China in 1996. He received a Master of Science degree in computer science and technology from Tsinghua Univ in 1998.

Publication

1. Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla S. Ellis. Power Aware Page Allocation. *In Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS IX)*, pages 105 116, November 2000.

2.Heng Zeng, Carla Ellis, Alvin Lebeck and Amin Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource, *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct, 2002.

3. Heng Zeng, Carla Ellis, Alvin Lebeck and Amin Vahdat. Currentcy: Unifying Policies for Resource Management, *USENIX 2003 Pp. 43-56 of the Proceedings*, June 2003.