ECOSystem: Managing Energy as a First Class Operating System Resource *

Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, Amin Vahdat
Department of Computer Science
Duke University
{zengh,carla,alvy,vahdat}@cs.duke.edu

Abstract

Energy consumption has recently been widely recognized as a major challenge of computer systems design. This paper explores how to support energy as a first-class operating system resource. Energy, because of its global system nature, presents challenges beyond those of conventional resource management. To meet these challenges we propose the Currentcy Model that unifies energy accounting over diverse hardware components and enables fair allocation of available energy among applications. Our particular goal is to extend battery lifetime by limiting the average discharge rate and to share this limited resource among competing tasks according to user preferences. To demonstrate how our framework supports explicit control over the battery resource we implemented ECOSystem, a modified Linux, that incorporates our currentcy model. Experimental results show that ECOSystem accurately accounts for the energy consumed by asynchronous device operation, can achieve a target battery lifetime, and proportionally shares the limited energy resource among competing tasks.

1. INTRODUCTION

Traditionally, the operating system plays two important roles. First, it provides a convenient set of abstractions of the raw hardware devices to application programs (e.g., a filesystem on top of a raw disk, virtual memory from physical memory). Second, the operating system allocates available system resources among competing applications to achieve target goals such as fairness or relative prioritization.

Today, available energy, as embodied by the system battery, plays an increasingly important role in the utility of many computing environments, from laptops and PDAs to emerging platforms such as wireless sensor networks. Despite wide-spread recognition of the importance of energy, operating systems currently do not provide application developers a convenient abstraction of the energy resource. There have been broad efforts to better manage the energy use of individual devices, for example, CPU voltage scaling [24, 25, 13, 33, 12, 9, 27, 26], disk spindown policies [21, 7, 6, 19, 14], power aware page allocation [20, 5], and energy-aware networking [18, 31]. However, there has been relatively little attention to managing energy as a first-class system resource and explicitly allocating it among competing applications.

Thus, the goal of this work is to develop a unifying set of abstractions for managing existing system resources under the umbrella of energy. One of the major contributions of our work is the introduction of an energy accounting framework based on a *currentcy model*[1] that unifies resource management for different components of the system and allows energy itself to be explicitly managed. Unifying resource management has often been mentioned as a desirable goal, but a focus on energy provides a compelling motivation to seriously pursue this idea. Energy has a global impact on all the components of any hardware base. In our framework, applications can spend their share of energy on processing, on disk I/O, or on network communication - with expenditures on different hardware components represented by a common model. A unified model makes energy use tradeoffs among hardware components explicit.

In general, there are two problems to consider at the OS-level for addressing specific energy-related goals. The first is to develop resource management policies that eliminate waste or overhead and make using the device as energy efficient as possible. An example is a disk spindown policy that uses the minimal energy whenever the disk is idle. This traditional approach to power management has typically been employed in a piecemeal, per-device fashion. We believe our currentcy model provides a framework to view such algorithms from a more systemwide perspective. The second approach is to change the offered workload to reduce the amount of work to be done. This is the underlying strategy in application adaptation where the amount of work is reduced, often by changing the fidelity of objects accessed, presumably in an undetectable or acceptably degraded manner for the application user [10]. One of the strengths of our approach is that, while providing abstractions that can facilitate such application involvement, it can also accommodate unmodified applications without requiring them all to be rewritten to become energy-aware. Without relying on application-based knowledge, other ways of reducing workload demands must be found. Our currentcy model provides a framework to express policies that selectively degrade the level of service to preserve en-

*This work is supported in part by the National Science Foundation (EIA-99772879,ITR-0082914,CCR-0204367), Intel, and Microsoft. Vahdat is also supported by an NSF CAREER award (CCR-9984328). Additional information on this work is available at `http://www.cs.duke.edu/ari/millywatt/`.

ASPLOS X, 10/02, San Jose, CA, USA.

[1] Currentcy is a coined term, combining the concepts of current (i.e., amps) and currency (i.e., $).

ergy capacity for more important work.

Observing that battery lifetime can be expanded by limiting the discharge rate [22, 28], we consider mechanisms to enable the development of energy management policies for controlling the discharge rate to meet a specified battery lifetime goal. The first level allocation decision is to determine how much currentcy can be allocated to all the active tasks in the next time interval so as to throttle to some target discharge rate. Essentially, this first-level allocation determines the ratio of active work that can be accomplished to enforced idleness that offers opportunities to power down components. Then, the second level decision is to proportionally share this allocation among competing tasks.

We have implemented an OS prototype—called ECOSystem—incorporating these energy allocation and accounting mechanisms. Experiments demonstrate that the system accurately accounts for asynchronous device operation and that the overall energy allocation can achieve a target battery lifetime. Furthermore, we show that simple policies for proportional sharing serve to distribute the performance impact of limiting the average discharge rate among competing tasks in a user-specified manner.

The remainder of this paper is organized as follows. In the next section, we outline the underlying assumptions of this work. In Section 3, we present the currentcy model and the design of the currentcy allocator. In Section 4, we describe the prototype implementation and in Section 5, we present the results of experiments to assess the benefits of this approach. Section 6 discusses areas where architectural enhancements could simplify our design. We discuss related work in Section 7 and then conclude.

2. BACKGROUND AND MOTIVATION

2.1 Battery Characteristics

Battery lifetime is an increasingly important performance metric, encompassing conventional mobile computing and emerging systems, such as distributed wireless sensor networks. Typically, users and system designers face a tradeoff between maximizing lifetime and traditional performance measures such as throughput and response time. Depending on the scenario, the goal might be to have the battery last just long enough to accomplish a specified task (e.g., finish the scheduled presentation on the way to the meeting) or a fixed amount of work (e.g., viewing a DVD movie). Thus, metrics have been proposed to capture the tradeoff between the work completed and battery lifetime [23]. Alternatively, the work might not be fixed, but an acceptable quality of service is desired for as long as possible (e.g., signal processing on a sensor node and ad hoc routing for other nodes).

Fortunately, simple battery models, such as Peukert's equation [22, 28], are available that adequately relate battery lifetime to factors we can control. Specifically, given parameter values describing the particular battery technology in use, we can calculate a limit on the current or power consumption that, if enforced, should result in reaching a target battery lifetime. In trying to achieve a given discharge rate, the first challenge is to accurately determine the level of resource consumption for all subcomponents over time.

One recent development in the OS-directed management of the battery resource is the *Smart Battery* interface in the ACPI specifications [15] and compatible battery devices that support it. This interface allows the system to query the status of the battery, including the present remaining capacity, present drain rate, and voltage. The Smart Battery seems to be a potentially powerful tool in support of energy management. However, our investigations of existing Smart Battery capabilities reveals limitations for our purposes. The operation of querying the interface is too slow to be useful for gathering power consumption data at a sufficiently fine grain for resource management functions such as scheduling without introducing unacceptable overhead. In addition, the averaging of power consumption data returned by the query makes attributing an accurate power consumption value to a particular process problematic, even with only the CPU involved. We ran experiments with two synthetic benchmarks that individually produce a distinct, stable power consumption profile showing that when they are scheduled together, the reported power values cannot be differentiated between the two competing processes. Even if better battery-related data were available, other accounting issues would remain related to accurately attributing power/energy usage, as described in the next subsection.

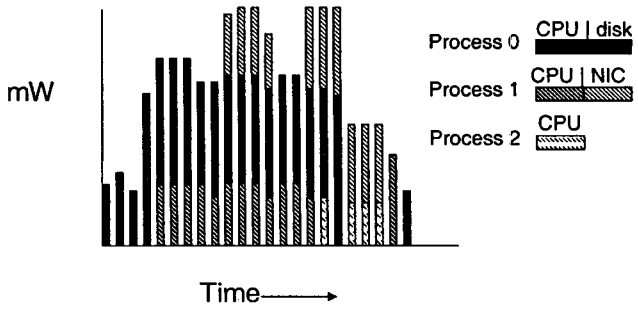

Figure 1: Accounting challenges of multiple devices and processes

2.2 Energy Resource Accounting Challenges

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. This motivates a unified model that can be used to characterize the power/energy consumption of all of these components. 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 accounting challenges. Consider the scenario portrayed in Figure 1 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 [11] or energy consumption [3], would inaccurately attribute power costs to the wrong processes.

Solving the accounting problem is a prerequisite to managing the battery resource. This involves (1) understanding the nature and determining the level of resource consumption, (2) appropriately charging for use of the various devices in the system, and (3) attributing these charges to the responsible entity. We introduce the

currentcy model to coherently charge for the energy consumption of many asynchronously active devices and we adapt *resource containers* [1] to serve as the abstraction to which energy expenditures are charged. The following section elaborates on our unified energy model.

3. THE CURRENTCY MODEL

The key feature of our model is the use of a common unit—*currentcy*—for energy accounting and allocation across a variety of hardware components and tasks. 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.

One unit of currentcy represents the right to consume a certain amount of energy within a fixed amount of time. The subtle difference between a unit of currentcy and a guarantee for an equivalent x Joules of energy is a time limit on use of the currentcy. This has the desired effect of pacing consumption.

Incorporating a generalized energy accounting model within the operating system provides the flexibility necessary to uniformly support a range of devices. The model can be parameterized according to the specific power characteristics of the host platform. With existing hardware support, there is no alternative that can provide the information about the power consumption of individual components needed for accounting. A side-effect of embedding this model in the system is that it also makes it possible to vary assumptions about the system's power budget to emulate alternative device characteristics. Thus, while our target environment uses energy in a certain fashion, we can also design experiments based on the profile of a PDA where the CPU and display power costs are significantly reduced and the hard drive may be eliminated altogether.

The remainder of this section describes the overall structure of our energy model and how currentcy can be credited to tasks and debited upon resource use to achieve a given battery lifetime.

3.1 System Energy Model

The system power costs are characterized in two parts of our model: The first part is the *base* power 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. While our experimental prototype with 3 managed devices (i.e., the CPU, disk, and network) is adequate to demonstrate our ability to unify multiple components under the currentcy model, the base remains a large, static factor in the range of discharge rates we are able to produce on the laptop. Thus, we are interested in investigating how changing this aspect of the power budget may affect the behavior of the energy allocation strategies we propose.

The second part of the system model is the specification of the costs of the more active states for each of the explicitly managed devices. Thus, the halted state of the CPU and the spun-down state of the disk fall into the base while CPU activity and spinning the disk are explicitly modeled. Each of these higher power states is represented by a charge policy that specifies how currentcy is to be deducted to pay for use of the target component.

The level of detail in this part of the model depends on the information that is available to the OS and the management choices available to it. The status of the device must be visible to the OS—either in terms of state information or as observable transition events that cause higher power use—to allow tracking of the state. Our current prototype is very coarse-grained (e.g., CPU halted or active) but the model can support finer-grain information such as using event counters to track processor behavior as suggested by Bellosa [2]. Our system could also benefit from component specific "gas gauges" that accurately measure energy consumption.

3.2 Currentcy Allocation

Our overall goal is to achieve a user-specified battery lifetime by limiting the discharge rate. There are two facets to the allocation strategy. The first level allocation determines how much currentcy can be made available collectively to all tasks in the system. We divide time into energy-epochs. At the start of each epoch, the system allocates a specific total amount of currentcy. The amount is determined by the discharge rate necessary to achieve the target battery lifetime according to our battery formula. By distributing less than 100% of the currentcy required to drive a fully active system during the epoch, components are idled or throttled. There are constraints on the accumulation of unspent currentcy so that epochs of low demand do not amass a wealth of currentcy that could result in very high peaks in the future. The second aspect of currentcy allocation is its distribution among competing tasks. When the available currentcy is limited, it is divided among the competing tasks according to user-specified proportions. During each epoch, an allowance is granted to each task according to its specified proportional share of currentcy.

Our model utilizes resource containers [1] to capture the activity of an application or task as it consumes energy throughout the system. Resource containers are the abstraction to which currentcy allocations are granted and the entities to be charged for energy consumed by the devices they use. They are also the basis for proportional sharing of available energy. Resource containers deal with variations in program structure that typically complicate accounting. For example, an application constructed of multiple processes can be represented by a single resource container for the purposes of energy accounting.

3.3 Currentcy Payback

Our initial resource management is based on a pay-as-you-go policy whereby a resource container gains access to a managed device. Consider the CPU – the process scheduler will allow ready processes to run as long as their associated resource containers have currentcy to pay for the time slice. When there are no processes whose resource containers have any remaining currentcy left, even though they may be otherwise ready to run, the processor is halted until the next allocation. Similarly, I/O operations that cause disk activity result in currentcy being deducted from the associated resource container. In this way, energy tradeoffs become explicit. Currentcy spent on I/O is no longer available to pay for CPU cycles.

Each managed device has its own charging policy that reflects the costs of the device. For example, the disk policy may try to spread out the payments for spinup or for spinning during the timeout period prior to spindown. The base costs are not explicitly

charged to resource containers, but obviously factor into the overall target power consumption. As we continue to develop the system, elements will migrate from the base into the category of explicitly managed and modeled devices.

The currentcy model provides the framework necessary to express a variety of energy-conscious policies ranging from CPU scheduling to disk management. Our implementation of one set of initial policies is described in Section 4. This allows us to demonstrate the feasibility of the currentcy model, to gain experience with the system, and to identify problems that motivate future research. We are actively exploring the rich design space of policies that can be formulated in the currentcy model [35].

4. PROTOTYPE

We implemented our currentcy model in the Linux operating system running on an IBM ThinkPad T20 laptop. This section describes our prototype implementation, called ECOSystem for the Energy-Centric Operating System. First, we provide a discussion of the specific power consumption values that are used to parameterize our model for the various hardware components in the T20. In Section 5.5, we examine the effects of changing these values to represent alternative platforms (e.g., PDA).

4.1 Platform Power Characteristics

We obtain the power characteristics of our Thinkpad hardware and use the resulting values as parameters to the currentcy model within the ECOSystem kernel. Within ECOSystem, we currently model three primary devices – CPU, disk, and network interface – by using microbenchmarks and measuring the actual power consumption with a Fluke multimeter. All other devices contribute to the base power consumption, measured to be 13W for the platform.

CPU

The CPU of our laptop is a 650MHz PIII. We use a coarse-grained abstraction that assumes that the CPU draws a fixed amount of power (we currently use 15.55 W) for computation. This was established by measuring the power while running a loop of integer operations. Ideally, one would like to charge differently for different processor behavior (e.g., various types of instructions or the frequency of cache misses, etc.) and this would be compatible with our modeling approach (e.g., by using event counters [2, 17]).

Disk

Many of today's hard disks support the ATA interface which uses a timeout based power management scheme to spin down an idle disk. The ATA standard defines a set of power states and the interface to control the timeout value for each state. Unfortunately, the hard disk in our laptop, an IBM Travelstar 12GN, has more power states than the ATA standard and these state transitions 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. Therefore, we approximate our disk's power consumption using a timeout based model derived from typical hard disks. Table 1 shows the values used in our model. It is well known that the energy cost to spinup the disk is high. In our case, we also observe that the power consumption increases when the IBM Travelstar tries to spin down the disk and we set the spindown cost to a fairly large value of 6000mJ. The disk model is set to spin down after 30 seconds. To achieve comparable effects on timing, we also set the Travelstar to spin down after 30 seconds.

	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 1: Hard disk power state and time-out values

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 (the default sleep time).

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 is 1500 bytes in our system.

4.2 The ECOSystem Implementation

We modified RedHat Linux version 2.4.0-test9 to incorporate energy as a first-class resource according to the model described in Section 3. Our changes include our own implementation of resource containers [1] to support the two dimensions of our model: energy allocation and energy accounting. Below we elaborate on the kernel modifications associated with each of these dimensions.

4.2.1 Currentcy Allocation

ECOSystem supports a simple interface to allow the user to manually set the target battery lifetime and to prioritize among competing tasks[2]. These values are translated into appropriate units for use with our currentcy model (one unit of currentcy is valued at 0.01mJ). The target battery lifetime is used to determine how much total currentcy can be allocated in each energy epoch. The task shares are used to distribute this available currentcy to the various tasks.

[2]We use the terms "task" and "resource container" interchangeably. One or more processes may comprise a task.

To perform the per-epoch currentcy allocation, we introduce a new kernel thread *kenrgd* that wakes up periodically and distributes currentcy appropriately. We empirically determine that a one second period for the energy epoch is sufficient to achieve smooth energy allocation. If a task does not use all its currentcy in an epoch, it can accumulate currentcy up to a maximum level (which is proportional to 10 times a task's per epoch share), beyond which any extra currentcy is discarded.

4.2.2 *Currentcy Accounting*

Tasks expend currentcy by executing on the CPU, performing disk accesses or sending/receiving messages through the network interface. The cost of these operations is deducted from the appropriate container. When the container is in debt (available-currentcy $\leq$ zero) none of the associated processes are scheduled or otherwise serviced. The remainder of this section explains how we perform energy accounting for the CPU, disk, and network card.

CPU

In our current implementation, a process is scheduled for execution only if its corresponding resource container has currentcy available. We modified the Linux scheduler to examine the appropriate resource container before scheduling a process for execution. Our CPU charging policy is based on a hybrid of sampling and standard task switch accounting. Accounting at a task switch provides accurate accounting of processor time used. However, to prevent long-running processes from continuing to run with insufficient currentcy, we deduct small charges as the task executes. If the task runs out of currentcy during its time-slice, it can be preempted early. Thus, we modify the timer interrupt handler to charge the interrupted task for the cost of executing one tick of the timer. In our system, a timer interrupt occurs every 10ms and the appropriate resource container's currentcy will be reduced by 15,550 units (155.5mJ). Under this policy, all tasks expend their currentcy as quickly as possible during a given energy epoch. This approach may produce bursty power consumption and irregular response times for some applications. We are currently developing a proportional scheduler that will more smoothly spread the currentcy expenditure throughout the entire energy epoch [35].

Hard Disk

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

To track disk energy consumption, we instrument file related system calls 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, we deduct an amount of currentcy from the appropriate resource container. Energy accounting for *read* operations is performed similarly.

We can break disk cost 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 our platform. Since a dirty buffer cache entry may not be flushed to disk for some time, multiple tasks may write to the same entry. Our 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, we believe 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. We assume 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 spindown). 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.

Our 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 our reported results, such activity constitutes only a small fraction of overall disk activity (as reflected by the overall accuracy of our achieved energy allocation).

Network Interface

Energy accounting for the network interface is implemented in ECOSystem by monitoring the number of bytes transmitted and received. 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 energy consumption is calculated at the device driver according to the full length of the packet including the header. We have instrumented the socket structure and the TCP/IP implementation 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. We believe that our approach with 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 our job of task tracking.

5. EXPERIMENTS AND RESULTS

This section presents experimental results using our prototype implementation. Our goal in this paper is to use ECOSystem to demonstrate that the unified currentcy model provides the framework necessary to manage energy as a first class resource. We begin by presenting our methodology. Then, we present a sequence of experiments designed to validate ECOSystem's energy accounting and allocation. A more extensive analysis of the policy design space created by the currentcy model is the focus of our ongoing research. We conclude this section by demonstrating the generality of the currentcy model by investigating alternative platforms.

5.1 Experimental Methodology

We use a combination of microbenchmarks and real applications to evaluate our framework. The microbenchmarks enable targeted evaluation of various system components. The real applications we use are netscape, xllamp, and ijpeg from the SPEC95 suite. The primary metrics are battery lifetime and application performance.

For each of our applications, we define an evaluation metric that we believe correlates with user-perceived performance. We measure application performance for various combinations of target lifetime and relative priorities. Our 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. We assume the page must be read from the network and that the netscape file cache is updated, so all three of our managed devices are included (CPU rendering, disk activity, and a network exchange). We obtain values for the performance metric by measuring the average duration of CPU activity for events (e.g., loading a web page). For netscape, we insert 5 seconds between page requests to model the user's think time. The think time has the effect of allowing some amount of currentcy to accumulate between events. Our next application, xllamp, is an MP3 player. This is representative of a popular battery-powered application with user-perceived quality constraints. Since each song has a specific play time, we evaluate this application's performance by measuring any slowdown in playback. This is done by comparing the actual time to complete the song against the length of the song. Any slowdown in playback manifests itself as disruption (e.g., silence) in the song. The final application, ijpeg, is computationally intensive and representative of digital image processing. The performance metric for ijpeg is execution time or processor utilization.

5.2 Energy Accounting

We begin by validating the energy accounting advantages of the unified currentcy model. For this experiment we use three synthetic benchmarks to exercise the disk, network 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 our benchmarks involved in wireless communication. The final benchmark is a CPU-only batch job (Compute) capable of running continuously. We execute the three benchmarks 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 1 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, we estimate that Compute, if run in a stand-alone fashion, should consume no more than 8,521,400 mJ (at 15.55 W).

We collect energy accounting data from ECOSystem's currentcy model. For comparison, we emulate a program counter sampling technique within ECOSystem by charging all disk and network activity to the resource container of the task occupying the CPU at each observation.

Table 2 shows the energy accounting results for both the unified currentcy model and the program counter based technique. Note that in the program counter sampling approach only the Total column would be reported, however we show 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.

5.3 Targeting Battery Lifetime

Achieving a target battery lifetime is an essential design objective of ECOSystem. Several potential sources of error exist in the energy accounting that could cause our behavior under the model to deviate from the target battery lifetime. For example, variations in cache behavior that are not captured by our flat CPU charge may introduce error in our lifetime estimate. One remedial approach involves making periodic corrections. By obtaining the remaining battery capacity via the smart battery interface once every 30 seconds, our system can take corrective action by changing the amount of currentcy allocated in subsequent energy epochs. If we appear to be under charging, then the overall currentcy allocation can be reduced. If it appears that we are over charging, then currentcy allocation can be increased.

To investigate the impact of energy accounting inaccuracies, we use our CPU intensive microbenchmark, but deliberately introduce accounting error for the CPU power consumption (14W instead of the measured 15.55W). Figure 2 shows the target battery lifetime for our platform on the x-axis and the achieved battery lifetime un-

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 2: Energy Accounting: Unified Currentcy Model vs. Program Counter Sampling

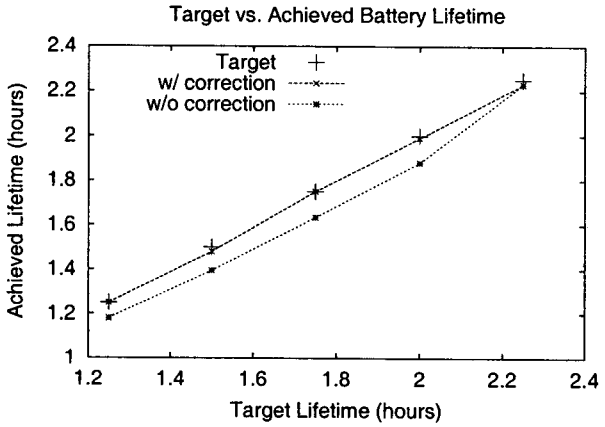

Figure 2: Achieving Target Battery Lifetime

der a variety of conditions 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 missing the target battery lifetime by approximately 10%. Finally, another curve on the graph shows that with our periodic corrections, we are able to achieve the target despite the deliberately introduced error.

5.4 Sharing Limited Energy Allocation

Having demonstrated the ability to achieve a target lifetime, the more important question involves proportionally allocating available energy among competing tasks. We validate the proportional sharing by running ijpeg and netscape simultaneously, and exploring a range of proportional currentcy allocations that represent user preference for interactive performance versus background computation performance.

At one extreme ijpeg is the higher priority task and receives the most currentcy. This represents a scenario where there is a computationally intensive application the user wants to run (e.g., image processing for a marketing presentation), but while it is running the user also wishes to browse the web[3]. At the other end of the spectrum, the user places higher priority on interactive performance.

Table 3 shows the effects of sharing limited energy resources. Our battery is rated at 3.6Ah and 10.8V. In this experiment we set the battery lifetime goal at 2.16 hours, obtaining an overall average discharge rate of 5W. Each row in the table represents a different partitioning of this 5W between the two tasks. If the tasks ran unconstrained they would draw approximately 16W and the battery would last only 1.3 hours. Note that with a 5W total allocation and a 15.55W processor, ijpeg would only receive 30% of the CPU even if it were allocated 100% of system energy.

Our results show that, as we modify the allocation of energy between the two tasks, each task closely matches the target allocation. While ijpeg is largely CPU bound, netscape does make use of all three modeled system components, CPU, disk, and the network. ECOSystem is able to match netscape's total power consumption across these three devices to the target power allocation. It is also interesting to note the non-linear relationship between power allocated to netscape and page load times. Doubling the power from 1.5W to 3.0W, reduces the page load time by approximately a factor of 4. Increasing power allocation by an additional 33% to 4W effects another factor of 2 reduction in average page load times. This result suggests a "knee in the curve" for interactive applications such as netscape, where additional power allocation beyond a certain point is unlikely to have a significant impact on the overall user-perceived "quality of service". For the CPU-intensive ijpeg application on the other hand, there is a direct linear relationship between the power allocated to the application and the amount of CPU time that it receives.

Our experiments thus far demonstrate that the unified currentcy model provides the framework necessary to: 1) accurately account for asynchronous energy consumption, 2) achieve a target battery lifetime by limiting the average discharge rate, and 3) proportionally share the limited currentcy allocation across competing tasks according to user specified allocations. However, an important aspect of any operating system abstraction is its ability to support new or different hardware devices.

5.5 Generality of the Currentcy Model

The currentcy model provides a powerful framework that can easily transfer to different hardware platforms. Supporting a different platform simply requires changing the model's device parameters. Adding new types of devices requires the appropriate kernel modifications to include the device in the resource container abstraction.

In this section, we utilize the flexibility of our 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 [4]). Recognizing this limitation, we wish to investigate a hypothetical platform that is representative of a future PDA-like device with a 2W processor, 0.02W base power, and MEMS-based storage. Our MEMS storage power characteristics are based on those presented by Schlosser et al [30]. An access costs 0.112mJ, transitioning to active mode costs

[3]This could also represent a sensor network scenario with a high priority computation (e.g. signal processing) and low-priority networking (e.g., ad hoc routing) task that should not dominate sensor network lifetime by unnecessarily draining the battery of a single node.

Energy Share	ijpeg			Netscape		
	Power Alloc(W)	Ave Power Used(W)	CPU Util(%)	Power Alloc(W)	Ave Power Used(W)	Page Load Latency(sec)
70%:30%	3.5	3.507	22.55%	1.5	1.49	29.205
60%:40%	3.0	3.008	19.34%	2.0	2.006	17.441
50%:50%	2.5	2.500	16.08%	2.5	2.457	9.928
40%:60%	2.0	2.008	12.91%	3.0	2.961	6.322
30%:70%	1.5	1.503	9.67%	3.5	3.443	3.934
20%:80%	1.0	1.005	6.46%	4.0	3.663	3.032

Table 3: Proportional Sharing: ijpeg vs. netscape, 5W Total Energy

5mJ and transitioning back to standby mode costs 0mJ. We assume the energy to remain active is 100mW, and the timeout to standby mode is 50ms. We use the same charging policy as in our hard disk model. For these experiments we use a 3.7Ah battery.

Consider the scenario where a PDA user is listening to music but also wishes to browse the web with the wireless network card. In this case the user wants the battery to last a specified amount of time and is willing to tolerate increased delays in web access to avoid annoying gaps in the song's playback. Note that this situation is similar to a sensor network scenario where there is a primary computation task (e.g., detecting a target) and a low priority task (e.g., ad hoc routing or data fusion).

Table 4 shows the results of sharing the available currentcy between the MP3 player (x11amp) and netscape for various amounts of available currentcy. X11amp requires 80mW to ensure continuous playback, the remaining capacity is available for netscape. These results show that as we increase the battery lifetime from about 9 hours (600mW allocation) to over 25 hours (200mW allocation) x11amp always receives its required energy, while netscape's portion decreases. The cost of this decreased energy allocation is an increase in response time from the minimum 3.12 seconds for a 9 hour lifetime to over 30 seconds for the 25 hour lifetime. In another test, if we run x11amp and ijpeg together unconstrained, these two applications would consume approximately 2.4W and the battery would last only 2.3 hours, but we can still achieve the uninterrupted playback and target battery lifetime with our approach (constraining ijpeg's CPU time appropriately).

These results show how the currentcy model can be modified to emulate 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.

6. SYSTEM ARCHITECTURE RECOMMENDATIONS

Our experience with ECOSystem exposed several opportunities for architectural support that could simplify or improve operating system development. As shown in the previous section, our approach of including a relatively simple model is sufficient to support managing energy as a first class resource. The recommendations provided below are primarily targeted at simplifying or streamlining the implementation.

The current model utilizes platform characteristics obtained through measurement of microbenchmarks exercising specific system components (e.g., disk, CPU, network). This makes it cumbersome to port the currentcy model to different platforms. To ease this transition, the platform architecture could provide support to monitor the energy consumption of each system component. For example, by exporting the appropriate interface, the disk controller could inform the operating system of the exact power consumed to perform specific accesses.

Operating systems would also benefit from obtaining precise information about the current state of a device. The ACPI interface dictates that all device power states are operating system controlled. However, the IBM disk drive used in our tests follows the philosophy that devices are better managed by local algorithms that exploit information not observable to the operating system. Unfortunately, often the internal algorithms are unknown to the operating system and thus the OS is unable to predict the disk behavior. We believe that if devices are locally managed they should provide information about their current power state so the operating system can incorporate that information into its policy decisions. An interesting area of future work is exploring collaborative disk power management.

It is possible that new operating system policies will increase the disk start/stop cycles, thus reducing reliability. In this case, it is imperative that disk manufacturers further increase the minimum allowable start/stop cycles. IBM's load/unload technology is an example of a technique to extend disk durability.

7. RELATED WORK

Attention to the issues of energy and power management is gaining momentum within operating systems research. Recent work has made the case for recognizing energy as a first-class resource to be explicitly managed by the operating system [8, 32].

Work by Flinn and Satyanarayanan on energy-aware adaptation using Odyssey [10] is closely related to our effort in several ways. Their approach 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.

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 [29], the operating system monitors processor temperature and when it reaches a threshold, the scheduling policy responds to limit

Target Lifetime(h)	Total Power (mW)	xllamp			Netscape		
		Power Alloc(mW)	Ave Power Used(mW)	Playback Time(sec)	Power Alloc(mW)	Ave Power Used(mW)	Page Load Latency(sec)
25.23	200	80	77	300	120	119	31.345
17.34	300	80	76	300	220	219	13.62
13.21	400	80	78	300	320	314	8.31
10.67	500	80	77	300	420	417	4.698
8.95	600	80	78	300	520	489	3.12

Table 4: Extending Battery Lifetime on an Alternative Platform: We assume a 2W CPU and MEMS disk, a base power of 0.02W, with a 3.7Ah battery and an operating voltage of 1.5v. The base power is 0.02W

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 our 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 [2]. 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.

The term "throttling" (which we have used in a very general sense) is most often associated with the growing literature on voltage/clock scheduling [24, 25, 13, 33, 12, 34, 16, 9, 27, 26] 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.

The body of literature on power/energy management has been dominated by consideration of individual components, in isolation, rather than taking a system-wide approach. Thus, in addition to the CPU-based studies mentioned above, there have been contributions addressing disk spindown policies [21, 7, 6, 19, 14], memory page allocation [20, 5], and wireless networking protocols [18, 31]. The emphasis is most of this work has been on dynamically managing the range of power states offered by the devices. This work is complementary to our currentcy model and will impact the charging policies for such devices in our framework.

8. SUMMARY AND CONCLUSIONS

The utility of emerging computing environments is increasingly limited by available energy rather than raw system performance. To date, there have been many efforts to limit the energy usage of specific hardware devices. We believe, however, that energy must 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, we propose a unifying abstraction to integrate power management into the two traditional tasks of the operating system, hardware abstraction and 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.

We offer the following contributions to this emerging research field. First, we propose a Currentcy Model that unifies diverse hardware resources under a single management framework and demonstrate its applicability to a variety of platforms. Next, we implement a prototype energy-centric operating system, ECOSystem, that incorporates our model and demonstrates techniques for explicit energy management with a total system point of view. We apply this system toward the specific problem of proportionally allocating resources among competing applications. Our framework provides a testbed for formulating various resource management policies in terms of currentcy. Finally, we use our framework to explore the complex interactions of energy conservation, allocation, and performance by running experiments with real and synthetic benchmarks on our prototype.

9. REFERENCES

[1] G. Banga, P. Druschel, and J. C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Third Symposium on Operating Systems Design and Implementation*, February 1999.

[2] F. Bellosa. The Benefits of Event-Driven Accounting in Power-Sensitive Systems. In *Proceedings of the SIGOPS European Workshop*, September 2000.

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

[4] L. T. Clark, E. Hoffman, J. Miller, M. Biyani, L. Luyun, S. Strazdus, M. Morrow, K. Velarde, and M. A. Yarch. An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications. *IEEE Journal of Solid-State Circuits*, 36(11):1599–1608, November 2001.

[5] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramiam, and M. 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.

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

[7] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the Power Hungry Disk. In *Proceedings of the 1994 Winter USENIX Conference*, pages 293–306, January 1994.

[8] 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.

[9] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance setting for dynamic voltage scaling. In *The Seventh Annual International Conference on Mobile Computing and Networking 2001*, pages 260–271, 2001.

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

[11] J. 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.

[12] K. Govil, E. Chan, and H. 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.

[13] 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.

[14] 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.

[15] Intel Corporation, Microsoft Corporation, and Toshiba Corporation. Advanced Configuration and Power Interface Specification. http://www.teleport.com/ acpi, December 1996.

[16] T. Ishihara and H. 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.

[17] R. Joseph and M. Martonosi. Run-Time Power Estimation in High Performance Microprocessors. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 135–140, August 2001.

[18] 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.

[19] 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.

[20] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power Aware Page Allocation. In *Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS IX)*, November 2000.

[21] K. Li, R. Kumpf, P. Horton, and T. Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *USENIX Association Winter Technical Conference Proceedings*, pages 279–291, 1994.

[22] D. Linden. *Handbook of Batteries*. McGraw Hill, 2nd edition, 1995.

[23] T. Martin and D. Siewiorek. A Power Metric for Mobile Systems. In *Proceedings of the 1996 International Symposium on Low Power Electronics and Design*, August 1996.

[24] 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.

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

[26] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th Symposium on Operating Systems Principles*, pages 89 – 102, October 2001.

[27] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *The Seventh Annual International Conference on Mobile Computing and Networking 2001*, pages 251–259, 2001.

[28] R. Powers. Batteries for low power electronics. *Proc. of the IEEE*, 83(4):687–693, April 1995.

[29] E. Rohou and M. Smith. Dynamically Managing Processor Temperature and Power. In *Proceedings of 2nd Workshop on Feedback Directed Optimization*, November 1999.

[30] 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.

[31] M. Stemm and R. 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.

[32] A. Vahdat, C. Ellis, and A. Lebeck. Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency. In *Proceedings of the 9th ACM SIGOPS European Workshop*, September 2000.

[33] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of First Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994.

[34] 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.

[35] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. Currentcy: Unifying Policies for Resource Management. Technical Report CS-2002-09, Duke University Computer Science, May 2002.