Ghosts in the Machine: Interfaces for Better Power Management

Manish Anand, Edmund B. Nightingale, and Jason Flinn
Department of Electrical Engineering and Computer Science
University of Michigan

ABSTRACT

We observe that the modularity of current power management algorithms often leads to poor results. We propose two new interfaces that pierce the abstraction barrier that inhibits device power management. First, an OS power manager allows applications to query the current power mode of I/O devices to evaluate the performance and energy cost of alternative strategies for reading and writing data. Second, we allow applications to disclose ghost hints *that enable better power management in the presence of multiple devices. Adaptive applications issue ghost hints to device power managers when they are forced to use a poor I/O path because a device is not in an ideal power mode; such hints allow devices to implement proactive power management strategies that do not depend upon passive load observation. Using these new interfaces, we implement a middleware layer that supports adaptive disk cache management. On an iPAQ handheld running Linux, our cache manager reduces interactive response time for a Web browser by 27% and decreases total energy usage by 9%. For a mail reader, the cache manager decreases response time by 42% and energy use by 5%.*

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management—*storage hierarchies*; D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.8 [**Operating Systems**]: Performance

General Terms

Management, Performance

Keywords

Power management, energy-awareness, adaptive caching

1. INTRODUCTION

Most I/O devices currently implement power management algorithms that do not consider the context in which they are operating. Each device attempts to optimize its own energy usage, not that of the entire mobile computer. Similarly, applications rarely take the power mode of I/O devices into account when deciding when and where to read or store data. This modularity leads to simplicity of implementation for both parties, but sacrifices both performance and energy conservation.

The goal of our work is to enable better power management by exposing additional context to applications and devices. For applications, we add an OS power manager that presents a common interface for querying device performance and energy characteristics. The power manager also exposes dynamic information that includes the power mode that is currently employed by each device. Using this context, adaptive applications can modify when and where they read and write data in order to save power.

For power management algorithms, we add a *ghost hint* interface that exposes "accesses that might have been." When an adaptive application chooses to not use a device because it is in an inappropriate power mode, it discloses to the device's power manager a ghost hint that quantifies the lost opportunity. The power manager uses ghost hints to proactively transition the device to power modes that better meet application needs. We show that such hints are often a vital part of an adaptive power management strategy; without ghost hints, some applications achieve much poorer performance or energy conservation.

To explore the effectiveness of these new interfaces, we have built a middleware layer that provides energy-aware data caching. We observe that remote access to data is fundamental to mobile computing. To guard against periods of disconnection or poor wireless network coverage, many mobile applications cache data on local disk. For this reason, local and remote copies of data will often exist simultaneously, implying that an adaptive application has an opportunity to save time and energy by dynamically deciding from which location it will read data. We have modified a Web browser and e-mail reader to use this cache — our results show substantial reduction in interactive response time for both applications, along with moderate energy savings.

A secondary goal of our work is to provide the user with a more intuitive power management interface. Currently, each device exposes a separate set of custom controls for tuning power management. These controls are typically not expressed in terms that are meaningful to the average user. We remedy this problem by providing a single, global knob that controls the power management of multiple I/O devices. This knob expresses the user's relative priorities for

Hitachi Microdrive		Cisco 350	
Mode	Power	Mode	Power
Performance Idle	760 mW	CAM	1410 mW
Low-Power Idle	340 mW	PSM	390 mW
Standby	80 mW		

This figure shows the power usage of the Hitachi 1GB microdrive and Cisco 350 802.11 wireless network interface in each supported power mode. Measurements were taken during idle periods when no reads or writes were occurring. We report the mean of five measurements — the standard deviation of all measurements is within 4%.

Figure 1: Power usage of two I/O devices

Hitachi Microdrive			
From Mode	To Mode	Time	Energy
Low Power Idle	Perf. Idle	300 ms	310 mJ
Standby	Perf. Idle	820 ms	900 mJ
Perf. Idle	Low Power Idle	160 ms	150 mJ
Perf. Idle	Standby	320 ms	300 mJ
Cisco 350			
From Mode	To Mode	Time	Energy
PSM	CAM	400 ms	510 mJ
CAM	PSM	410 ms	530 mJ

This figure shows the cost of power mode transitions for the Hitachi 1GB microdrive and Cisco 350 802.11 wireless network interface. We report the mean of five measurements — the standard deviation of all measurements is within 7%.

Figure 2: Device transition costs

performance and energy conservation. We show that it is possible to translate this global value into specific strategies for dynamically tuning custom device power management controls by inserting a self-tuning software layer.

We begin in the next section by describing current power management and its limitations in more detail. Section 3 outlines our design goals and Section 4 discusses our implementation. Section 5 describes out experimental results. We conclude with a discussion of related and future work.

2. BACKGROUND AND MOTIVATION

2.1 Dynamic power management

In the absence of power management, the energy used by I/O devices can be prohibitive for small, mobile computers. For example, Figure 1 details the power usage of the Cisco 350 802.11 wireless network interface and the 1 GB Hitachi microdrive. Without power management (in CAM and performance idle modes, respectively), these devices draw a combined 2.17 Watts of power without any I/O activity. Given that the rest of the iPAQ handheld draws only 1.44 Watts when idle, a back-of-the-envelope calculation shows that using these two devices without power management decreases battery lifetime by 60%.

For this reason, most I/O devices intended for mobile computers support one or more power management modes. The microdrive's low-power idle mode reduces its power consumption by 55% by shutting down many electronic components. However, the next access incurs a performance penalty of approximately 300 ms because the drive must transition back to performance idle before accessing the disk. Similarly, the microdrive's standby mode decreases power consumption by almost 90%, but the next access incurs a transition cost of approximately 800 ms. The Cisco 802.11 card supports a power-saving mode (PSM) that disables the network interface if no incoming packets are waiting for transmission at the wireless access point. The interface wakes up periodically (typically every 100 ms) to poll the base station for new packet arrivals. This mode reduces power consumption by 72%; however, incoming packets may incur a delay proportional to the polling period, leading to high latency and reduced throughput for network communication. Figure 2 further details the transition cost of changing power modes.

Both devices implement dynamic power management algorithms that adaptively switch between the supported power modes. Such algorithms observe device access patterns to determine when to change modes. They attempt to use high-performance modes during periods of high activity and power-saving modes during idle periods.

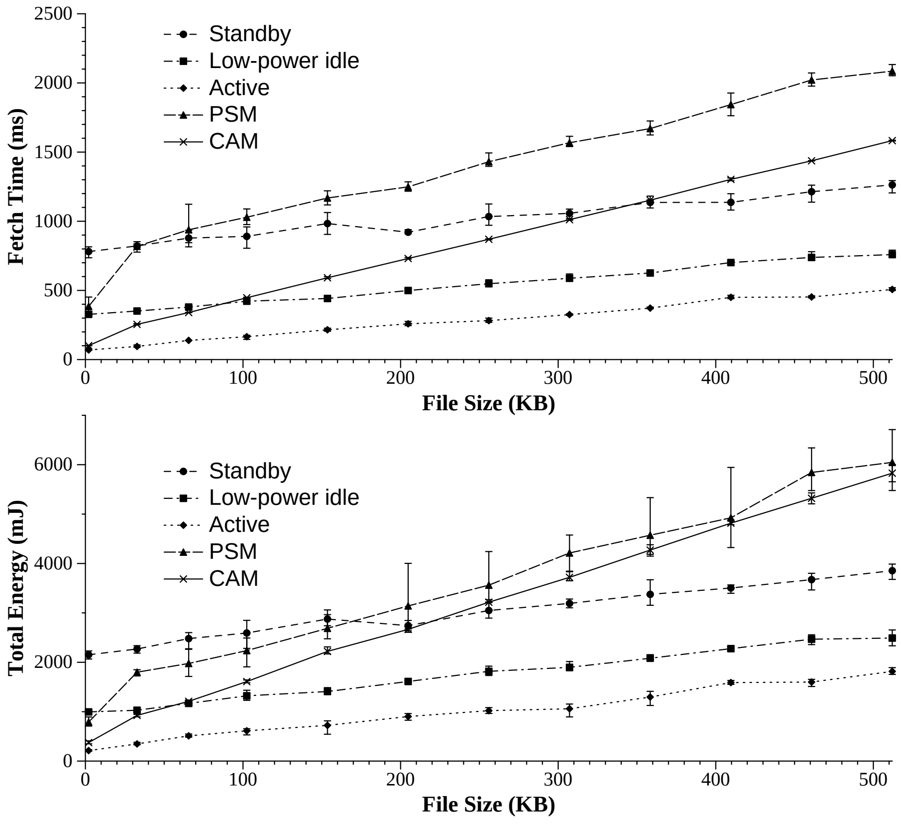
2.2 Impact of power management

It seems intuitive that fetching data from local storage should be less costly than fetching data from a remote server. In the absence of power management, this is usually true. However, as Figure 3 shows, the time and energy needed to read data from network and storage devices is severely impacted by the power modes being employed. To generate these results, we ran the Dillo Web browser [6] on an HP iPAQ handheld with a Cisco 350 wireless interface and and a 1 GB Hitachi microdrive. Dillo does not natively provide disk caching, so we used the wwwoffle disk caching proxy [3] to read and write to local storage. We used NISTnet [4] to emulate a high-speed cable modem connection; we added a 20 ms latency between the client and Web server, as well as an upstream cap on bandwidth of 250 Kb/s and a downstream cap of 3 Mb/s.

The results show that reading data from local storage is best from a time and energy perspective *if the local disk is in a high-power mode.* However, if the microdrive is currently in a power-saving mode, it takes less time and energy to fetch a small object from the network server. Reading a 32 KB file from the microdrive takes 820 ms and 2.3 Joules if the drive is in standby mode, but reading the file from the server requires only 250 ms and 0.9 Joules if the network is in CAM. For large objects, the most efficient method of reading data is to pay the transition cost of spinning up the microdrive and then read the object off disk. Reading a 512 KB file from the drive in standby mode takes 320 ms less than reading it from the network in CAM. While variation in available network bandwidth and latency will change the break-even point for these results, a network connection must be quite poor to exceed the 800 ms cost of spinning up the hard drive.

2.3 The benefit (and limitations) of adaptation

Many applications cache data on mobile computers to guard against disconnection or poor wireless connectivity. Examples include Web browsers [3, 6, 22], e-mail applications [7], and distributed file systems [16]. Further, many mobile computers have multiple storage or network options; for example Bluetooth and 802.11. An adaptive application can improve both performance and energy conservation by



The top graph shows the time to read Web data from a 1 GB Hitachi microdrive and from a Web server using a Cisco 350 wireless network interface. The bottom graph shows energy usage. The standby, low-power idle and active lines show time and energy used to fetch data from disk with the drive initially in each mode. The CAM and PSM lines show time and energy to fetch data from the network with the network interface in each mode. Each data point is the mean of three trials; the error bars show the minimum and maximum trials.

Figure 3: Effect of network and storage power modes on Web fetch time and energy

observing the power modes of its network and storage devices and dynamically deciding from which device it will fetch each data object. Such an application could, for instance, fetch small items from the network when the disk is in standby mode. In contrast, current mobile application embed a fixed cache hierarchy—they fetch an item from disk even if it is cheaper to obtain the data over the network.

One relevant question is whether mobile devices in the future will eschew disk drives entirely for reasons of energy-efficiency. Currently, disk technology offers greater capacity and is generally more cost effective than flash memory. While prices change rapidly, as of November, 2003, a 4 GB Hitachi microdrive was priced at $489, a 1 GB microdrive was $185, and a 1 GB SanDisk Compact Flash card was $249 (we could find no 4 GB flash product offered) [27]. Further, this work shows that it is possible to achieve reasonable performance using disk technology with minimal impact on battery lifetime.

Finally, a careful study of the results in Figure 3 reveals that adaptation, by itself, is insufficient to achieve the best possible performance improvement. Consider a Web application that fetches many small images stored in the disk cache. If the disk is in standby mode, the adaptive application would fetch each image from the network server since the cost of network access for each item is smaller than the time and energy cost of spinning up the hard drive. For each individual access, this decision is correct. Yet, if the drive were to transition to a high-power mode, the transition cost would be amortized across many reads, leading to significant energy and performance savings. Unfortunately, such behavior is infeasible with current power management strategies that base their decision upon observed accesses — since the drive is never accessed, the power management algorithm will never transition to a high-performance mode.

Our approach to solving this problem is the addition of *ghost hints* that describe "accesses that might have been". The adaptive application first calculates the best possible method for accessing data given each device's current power state. It fetches the data using this method. However, the application also computes what method would have been best if all devices had been in the ideal power mode. If this ideal method differs from the method actually used, the application issues a ghost hint to the power management layer that describes the opportunity lost due to a device not being in the ideal power mode.

In the above example, the disk power manager would receive several ghost hints and respond by spinning up the disk. Thus, the application would fetch the first few files from the network until the disk has completed its transition; once the disk is spinning, subsequent accesses would

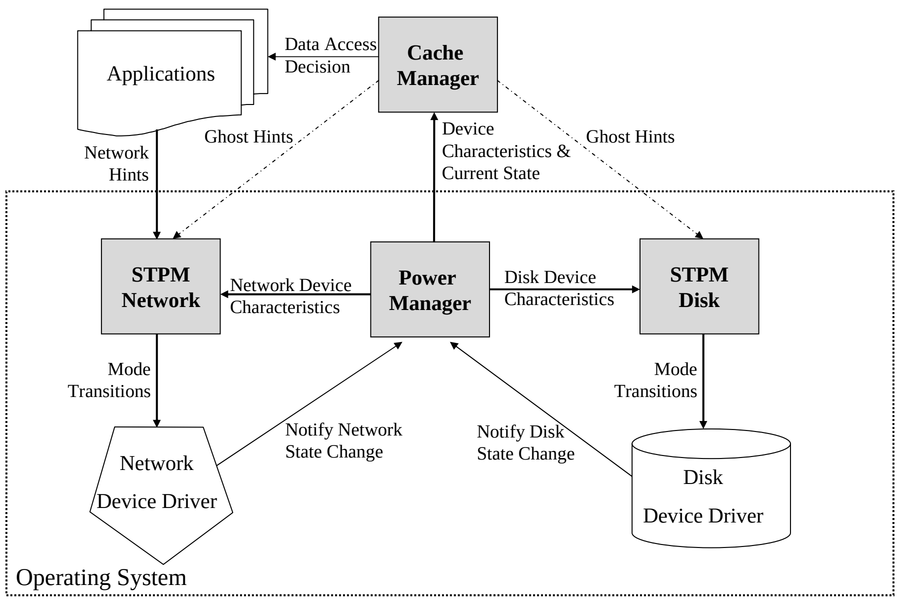


Figure 4: Energy-aware architecture

go to disk, saving both time and energy. Further, since the network device would no longer be used by the application, the network power manager would transition to a low-power mode. No ghost hints are issued to the network at this point because the disk offers the best possible performance while it is spinning.

3. DESIGN GOALS

Our primary design goal is to provide simple interfaces that enable better power management by piercing the abstraction barrier that currently separates applications and the power management layer. In essence, this barrier is yet another instance of the familiar tension between performance and modularity in system design [19]. Our belief is that current power management algorithms sacrifice too much to achieve modularity — by exposing a small amount of additional context, we gain substantial improvement in performance and energy conservation.

Currently, almost all mobile computing devices implement a custom power management algorithm in isolation. For example, the Hitachi microdrive decides when to enter power-saving modes by observing the pattern of recent disk accesses [14], and the Cisco 350 802.11 network interface adaptively switches between PSM and CAM depending upon traffic load [5]. These algorithms do not consider context such as the base power of the mobile computer or application intent. This often leads to decisions that are inferior to those that could be made by a more informed power management algorithm. Further, each device typically exports a custom set of controls for tuning its custom power management algorithm. This leads to an explosion of complexity: Windows XP on a typical laptop has over 25 separate knobs that can be used to adjust power management behavior. A secondary goal of our work is to reduce this complexity by providing a global control that tunes the power management of all devices simultaneously. While more work, including a human factors study, is needed to evaluate this control, the interfaces developed here represent a necessary first step in this direction.

4. IMPLEMENTATION

Figure 4 shows our energy-aware caching architecture. The power manager, described in the next section, is the central repository for static and dynamic data describing device power modes and their associated performance and energy characteristics. The power manager is currently implemented as a Linux loadable kernel module.

The second component of our architecture is self-tuning power management (STPM), described in Section 4.2. STPM algorithms consider more than just device access patterns: they dynamically adjust their behavior to the base power of the mobile computer as well as the relative priority of performance and energy conservation. We implement self-tuning power management in the kernel at a layer above individual device drivers. All STPM modules expose a simple common interface. This enables users to adjust the behavior of all devices simultaneously using a global knob. Each STPM module translates the knob setting into a device-specific policy for dynamically adjusting the custom controls exported by the particular device that it is managing.

An adaptive cache manager, described in Section 4.3, dynamically determines the best source from which to read data by querying the current power mode of available devices and predicting the performance and energy cost of reading from each source. The cache manager also provides delayed write mechanisms that amortize the cost of I/O access. We have modified an e-mail reader and a Web browser to use our cache manager.

The final component is an interface for *ghost hints* that is described in section 4.4. The cache manager uses ghost hints

```
RegisterDevice        (IN device, IN mode_data, IN transition_costs, OUT handle);
Notify                (IN handle, IN new_mode);
DeregisterDevice      (IN handle);
QueryNumModes         (IN device, OUT num_modes);
QueryModeInfo         (IN device, IN mode, OUT mode_info);
QueryTransitionCost   (IN device, IN from_mode, IN to_mode, OUT cost);
QueryCurrentMode      (IN device, OUT mode);
RegisterCallback      (IN device, IN mode);
```

Figure 5: Power manager interface

to inform STPM modules that it would have used a particular device *if only that device had been in another power mode*. Ghost hints enable devices to proactively transition to high-performance modes.

4.1 The OS power manager

The power manager provides a common interface that enables applications to query the performance and energy characteristics of I/O devices. Such information is a vital first step in the implementation of energy-aware strategies at user level. Without such an interface, applications would be unable to tune their behavior to the specific I/O devices present on a mobile computer. Further, applications would be unable to adjust their behavior in response to dynamic power management.

We considered an alternative strategy of having each device export a custom interface. While this strategy could potentially offer more flexibility, we feel that the power manager adds a useful layer of indirection. Implementation of energy-aware strategies is simplified because data about all I/O devices is obtained through a single interface. Device driver implementation is simplified because the power manager provides common functionality for notifying applications when power modes change.

Figure 5 shows the power manager interface. The first three functions are exported to device drivers; the next five are exported to applications. When a device is loaded, its driver calls `RegisterDevice` and discloses the device's supported power modes, as well as performance and energy characteristics in each mode. Whenever the device changes to a new power mode, the driver calls `Notify` to inform the power manager.

We have currently modified two Linux device drivers to use this interface: the wireless network driver used by the Cisco 350 802.11 network card, and the IDE driver used by the Hitachi microdrive. We derived the performance and energy characterization for each device, shown in Figures 1 and 2, by running a benchmark suite. This characterization must be performed only once for each type of device. Eventually, we hope that manufacturers and device driver developers can make such information a standard part of the driver interface since much of the necessary information is already provided as part of the device specifications. For our characterization, we used actual measurements, further described in [1], rather than the stated specifications (although we found few significant discrepancies).

The application interface allows user level programs to obtain static and dynamic data about the power modes supported by each device. `QueryModeInfo` returns attributes of a specified power mode such as the power drawn by the device and whether the device is capable of reading or writing data in that mode. Further, if the device can read or write data in the queried power mode, `QueryModeInfo` also returns a model of the performance and energy costs of I/O operations. `QueryTransitionCost` returns the cost of changing modes — this is expressed in terms of both time and energy.

`QueryCurrentMode` returns the power mode of a specified device. Applications may also use `RegisterCallback` to block until a device enters a specific mode. The callback interface is useful for applications that can delay activity and wish to avoid polling. For example, a Web browser might wish to delay writing to its disk cache until the disk is already spinning.

4.2 Self-tuning power management

STPM modules make better power management decisions because they consider more context than just device access patterns. Each module explicitly considers the base power of the mobile computer and exports an interface that allows the user to adjust the relative priority of performance and energy conservation.

When a device is inserted into a laptop with high base power, its power management algorithm should be conservative. When it is inserted into a handheld with low base power, its power management algorithm should be more aggressive. The reason for this difference is that the benefit of power management, i.e. the relative percentage of energy saved, is much greater on the handheld. Further, the performance cost of power management is constant on both devices, but the laptop will expend more total energy waiting for I/O accesses since its base power is higher. In previous work [1], we have found that using default (i.e. untuned) wireless network power management on laptops can actually *increase* the energy used to perform common workloads.

The priority of performance and energy conservation is exported to the user as a global knob that ranges in value between 0 and 100. Each STPM module considers the value of this knob in its power management decisions. When the knob is set to 0, each STPM module implements the power management policy that maximizes energy conservation. When the knob is set to 100, each module maximizes performance. Intermediary values represent different relative weightings of these two goals. This knob reflects the observation that the priority of these goals varies according to usage context. If a user expects to operate on battery power for only a few minutes, performance is the primary concern. The longer the user needs the battery to last, the more important energy conservation becomes.

The primary intent of this knob is to make power management more intuitive for the user. The user can control the activity of many devices with one simple interface. Further, this control is expressed in terms that are meaningful to the user, i.e. performance and energy conservation. In contrast, current devices export custom controls, meaning that a user must adjust many different knobs when greater performance or extended battery lifetime is needed. Additionally, these controls are often expressed in terms such as CAM, low-power idle, etc. that are unclear to the average user. It is non-trivial to predict how adjusting these controls will affect performance and energy conservation.

Finally, STPM modules allow applications to supply additional context about device accesses. For instance, an application may hint that a particular access is a background request for which performance is not critical. A distributed file system might use such hints to specify that prefetch requests and delayed write operations are background traffic.

4.2.1 Network power management

As our network self-tuning power management module has been described in detail elsewhere [1], we review here only the relevant properties of this module. Our STPM network module presents a hinting interface that enables applications to disclose how groups of packets sent and received by the network interface are mapped to logical data transfers. For applications that do not disclose such hints, the module observes network traffic and uses heuristics to perform the mapping. The module groups transfers into *runs* that represent sets of transfers that are closely correlated together in time. It maintains histograms of run length and inter-arrival time between runs. Considering both the base power of the computer and the current relative priority of performance and energy conservation, it performs a cost-benefit analysis to calculate the best predicted policy for power management. This policy describes when the network device should transition to each power mode; for example, the STPM module might transition from PSM to CAM after it observes three consecutive transfers in a run. Our results show that STPM improves performance and reduces energy usage for applications such as distributed file systems, streaming audio, and remote X window display.

4.2.2 Disk power management

The disk STPM module decides when to transition to a low-power mode (e.g. low-power idle or standby for the microdrive) and when to transition back (e.g. to the performance idle mode). A common practice in disk power management is to use the *break-even time* as a heuristic for determining when to transition to a low-power mode [21, 28]. We adopt this heuristic as a starting point and then apply the STPM principles to improve it.

The break-even time is the amount of time that a disk must remain idle in order to save energy by transitioning to a low-power mode. As can be seen in Figure 2, the microdrive expends considerable energy during power mode transitions. Further, if the next disk request that is made after entering the low-power mode is a foreground activity (as is likely for most interactive applications), the mobile computer will expend additional energy while waiting for the request to be serviced. Thus, by knowing the amount of energy expended by the mobile computer during transitions, E_{up} and E_{down}, as well as the amount of power used while in the high-power and low-power modes, P_{hp} and P_{lp}, we can calculate the break-even time, T_{be}, as:

$$T_{be} = (E_{up} + E_{down})/(P_{hp} - P_{lp}) \qquad (1)$$

The break-even heuristic assumes that the disk is more likely to remain idle after it has already been idle for a period of time. It therefore transitions to a low-power mode after the disk has already been idle for the break-even time.

In our STPM module, we modify this heuristic to take into account the base power of the mobile computer and the relative priority for performance and energy conservation. First, we use base power to calculate energy expenditure for the entire computer, not just the disk device. Second, we consider both performance and energy conservation when deciding when to transition the disk and use the knob value to assign a relative weighting to each goal.

Specifically, the time cost of using a low-power mode is T_{up}, the performance hit we incur on the next request while waiting for the disk to transition to the high-power state. The energy cost, C_E, is calculated as:

$$C_E = E_{down} - (P_{hp} * T_{down}) + E_{up} + (P_{base} * T_{up}) \qquad (2)$$

During a transition to the low-power state, the mobile computer is idle, so we must consider only the extra energy expended by the disk to make the mode transition. During the reverse transition at the end of the idle period, the next request is delayed, forcing the entire mobile computer to waste power. We must therefore consider base power in this cost.

We weight time and energy cost according to the knob value, i.e. we treat K as the utility of reducing interactive delay by one second, and we treat $100 - K$ as the utility of saving a Joule. We calculate break-even time as:

$$T_{be} = \frac{(T_{up} * K) + (C_E * (100 - K))}{(P_{hp} - P_{lp}) * (100 - K)} \qquad (3)$$

The time and energy parameters, knob value, and base power for these calculations are provided by the OS power manager. Currently, the user sets the global knob value using a command-line tool. For the microdrive, the disk STPM module calculates the break-even point for low-power idle and standby transitions. During idle periods, it first transitions to low-power idle and then transitions to standby.

In general, current disk power management algorithms transition back to the high-power mode when the next request arrives. In our work, we consider two additional cases. First, background traffic offers the opportunity to delay the disk access until a more opportune time in the future. We explore this idea further in the next section. Second, a proactive disk power management policy may decide to transition to a high-power state even if no application is currently accessing the device. In section 4.4, we describe how the STPM module uses ghost hints to proactively transition the drive and provide adaptive applications with the opportunity for further power savings.

4.3 Middleware for energy-aware caching

Mobile applications such as Web browsers and e-mail clients often proactively cache files on local media to make data available during future periods of disconnection from the network or poor network connectivity. Such applications typically use a fixed cache hierarchy in which local storage is always considered cheaper to access than remote storage. However, as the results in Figure 3 show, device power management substantially impacts the cost of accessing data and can often make remote access cheaper than local storage.

```
CacheStatus    (IN name, OUT status, OUT size);
GetData        (IN name, OUT data);
GetMetadata    (IN name, OUT metadata);
PutData        (IN name, IN data);
PutMetadata    (IN name, IN metadata);
Rename         (IN new_name, IN old_name);
Delete         (IN name);
```

Figure 6: Cache manager interface

We chose energy-aware caching as the first implementation of our ideas because it allows us to restrict the problem domain to the simpler case of read-only data. In contrast, supporting a distributed file system such as Coda that modifies data locally requires careful handling to ensure data consistency. In this work, we further restrict our implementation to use a single network interface and a single storage device. Section 7 discusses our plans to extend our work into less restricted domains.

We chose a middleware approach to amortize the development cost of adding energy-awareness across multiple applications. While the development complexity of the caching layer is non-trivial, it exports a simple interface that can be easily added to existing applications.

The cache manager API is shown in Figure 6. When reading data, applications first call `CacheStatus` to determine whether a file is stored in the cache. This function returns one of three results: the file is not present, the file is present, or the file is present but it would be preferable to fetch the file from the network. This last result is advisory: the application may fetch the requested file from the cache if desired. The cache status also returns the size of the cached file.

The cache manager maintains an in-memory hash table with the name and the size of each item — this allows it to answer status queries without generating disk accesses. When an item is present, it calculates the most efficient manner of reading the data. At startup, it queries the OS power manager to determine the static performance and energy characteristics of network and storage devices. When it receives a status query for a cached item, it retrieves the current power mode of each device from the power manager. Given this data and the item size, it calculates the expected time and energy needed to read the item from network and from disk. It uses the knob value, K, to weigh these costs:

$$Cost_{total} = Exp\_time * K + Exp\_energy * (100 - K) \quad (4)$$

Depending upon the returned status code, an application either reads an item from disk using `GetData` or fetches the item from a network server. After fetching an item from the network, the application may call `PutData` to store the item in the cache. Since this is a background request, the cache manager defers the write when the disk is in a low-power state by placing the request on its write queue. A separate writeback thread registers for an OS power manager callback to be delivered when the disk next enters the performance idle state. After receiving the callback, it flushes the write queue to disk. We currently cap the maximum amount of data stored in the queue at 4 MB to alleviate memory concerns. This cap may be unnecessary, however, since memory pressure will lead to paging, and the associated disk activity will trigger a write queue flush. Queued data will be lost in the event of system crash — however, given that this is only a cache, we feel this disadvantage is more than outweighed by the potential energy savings. Alternatively, we could have used Weissel's Cooperative I/O interface [28] were it present on our machine. However, our user level implementation has the advantage of simplicity.

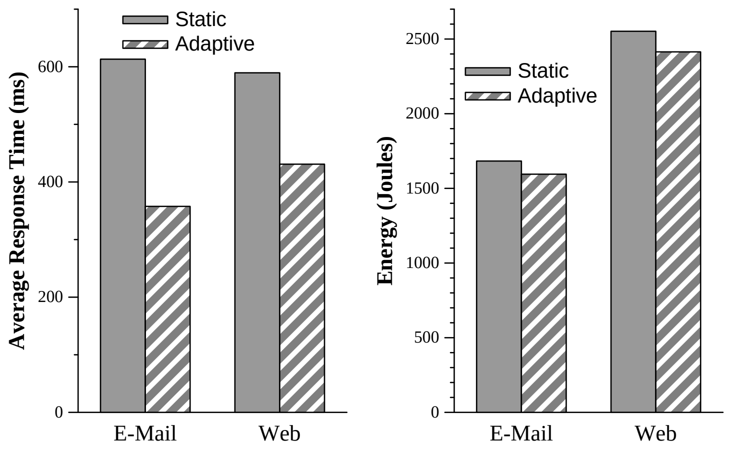
The `PutMetadata` call stores application-specific metadata for each object. Metadata items are kept in the in-memory hash table and are also written to disk using the deferred write mechanism. The Web browser uses this functionality to store HTTP header information. The remaining calls allow applications to rename and delete cached objects. These operations are also deferred in the write queue.

We have modified an e-mail client and Web browser to use our cache manager. In addition, we have changed these applications to disclose the start and end of network transfers to the network STPM module. These network hints enable better network power management and also provide the ability to implement simple estimation of network bandwidth and latency.

4.4 Ghost hints

Adaptivity alone may be insufficient to realize substantial performance or energy benefit across multiple devices. Although the adaptive cache may choose the best device at any given moment, it may lose considerable opportunity because one or more devices are not in their best possible power modes. For example, in Figure 3, if the disk is in standby mode and the network in CAM, an adaptive Web browser will ignore the disk cache and fetch a 32 KB item from the network saving 570 ms and 1,400 mJ. However, if the microdrive had been in performance idle mode, the Web browser would have read the cached copy, saving an additional 160 ms and 580 mJ. For a single object, it does not make sense to transition to a high-power mode, but if the Web browser is fetching many such items in short succession, the transition cost will be amortized across many reads, leading to overall performance and energy savings. Unfortunately, the disk power manager will never transition the device because it observes no accesses.

How can this dilemma be resolved? We see three possible solutions. The first is to implement a gray-box strategy [2] at user level that predicts future accesses and issues a foreground read to force the disk into a high-power mode. We rejected this approach because it is requires the user level process to have explicit knowledge of device power management strategies — this sacrifices far more modularity than we are currently proposing. Also, it is difficult to support multiple processes that use the same device with this approach. A second possibility is to implement a global operating system component that implements power management for all devices on a mobile computer. Global knowledge may certainly enable better power management, but may be too complex



The graph on the left compares the average response time achieved by our adaptive cache with that achieved by the default application caching strategy. The graph on the right compares the total energy needed to execute the trace.

Figure 7: Comparison of static and dynamic strategies

to implement. Further, this approach sacrifices even more modularity than the gray-box solution. Without application support, it might prove difficult to determine which accesses could be performed on which devices.

We call the solution we chose *ghost hints*. A ghost hint allows applications such as our cache manager to expose "accesses that might have been" to device power managers. When the adaptive cache manager chooses the device it will access, it also computes the time and energy that would have been needed to access alternative devices had they been in their highest power modes. If a device could have offered a better alternative for reading or writing data than the alternative actually chosen, the cache manager issues a ghost hint to the device power manager. In the above example, the Web browser will issue a ghost hint to the disk power manager for each 32 KB item that it fetches from the network. After receiving a few such hints in a short time period, the disk power manager transitions the drive to performance idle. During the transition, the Web browser continues to access the network, but subsequent accesses reap the benefit of the disk transition. Once the disk is in performance idle, the network may transition to PSM since it is no longer being used. Since the network cannot offer better performance in any power mode, no ghost hints are issued.

In each ghost hint, the cache manager specifies the potential time and energy that would have been saved had the device been in the ideal power mode. The disk STPM module uses this information to calculate the weighted benefit that would have been derived from being in the performance idle mode. It maintains a running total of such benefits. The total is increased every time a ghost hint is received — it is also decremented by the energy cost that would have occurred by remaining in the performance idle mode since the last ghost hint was issued. When this total exceeds the break-even threshold, calculated in Equation 3, the module transitions the microdrive to performance idle.

When the network STPM module receives a ghost hint, it increments the count of transfers in the current run. Several successive ghost hints cause a transition to CAM.

5. EVALUATION

We validate our power management architecture by answering the following questions:

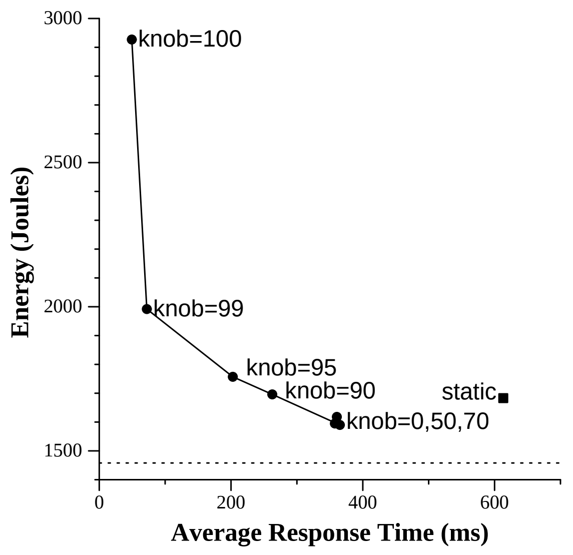
- Can adaptive, cross-device strategies deliver substantial performance improvements without sacrificing energy conservation?
- Can a global knob allow the user to adjust system behavior to meet different relative priorities for performance and energy conservation?
- How important are ghost hints for realizing the full benefits of cross-device power management?

5.1 Methodology

We use a HP iPAQ 3870 running the Linux 2.4.18-rmk3 kernel as the client platform in our evaluation. This handheld has a 206 MHz StrongArm processor, 64 MB of DRAM, and 32 MB of flash memory. We measure the base power usage of the iPAQ (with the display lit and the processor idle) at 1.44 Watts. We add a Cisco 350 802.11b network interface and a 1 GB Hitachi microdrive with power characteristics shown in Figures 1 and 2. The client communicates with a Cisco Aironet 350 wireless network access point.

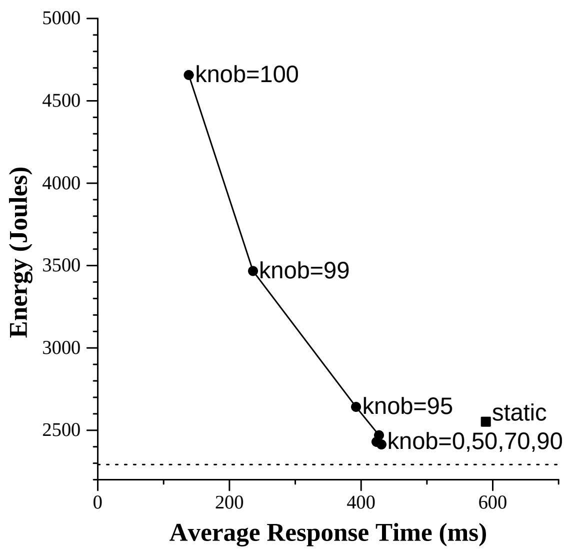
The server in our experiments is a Dell Precision 350 desktop with 3.06 GHz Pentium 4 processor and 1 GB DRAM running the Linux 2.4.18-19.8.0 kernel. Packets sent between the client and server are routed through an identical Dell desktop running the NISTnet [4] network emulator. We emulate a typical high-speed broadband connection in our area by adding a 20 ms delay to all packets (i.e. a 40 ms round-trip time), and by capping the upstream bandwidth at 256 Kb/s and the download bandwidth at 3 Mb/s.

For each experiment, we report average interactive response time and total energy usage. Response time is measured by instrumenting applications with the `gettimeofday` system call. Energy usage is measured with an Agilent 34401A digital multimeter. We remove the batteries from the iPAQ and its expansion sleeve and sample current drawn



This graph shows average response time and total energy used by the e-mail reader for seven different STPM knob values. Knob values from 0 to 70 yield equivalent results. For reference, the static disk cache strategy is also shown. The dotted line on the bottom graph is a lower bound on the minimum energy needed to execute the trace.

Figure 8: Performance/Energy tradeoff for e-mail



This graph shows average response time and total energy used by the Web browser for seven different STPM knob values. Knob values from 0 to 90 yield equivalent results. For reference, the static disk cache strategy is also shown. The dotted line on the bottom graph is a lower bound on the minimum energy needed to execute the trace.

Figure 9: Performance/Energy tradeoff for Web

through the iPAQ's external power supply approximately 50 times per second. We calculate system power usage by multiplying each current sample by the mean voltage drawn by the iPAQ—separate voltage samples are not necessary since the variation in voltage drawn through the external power supply is very small. We calculate total energy usage by multiplying the average power drawn during benchmark execution by the time needed to complete execution.

We first tested our cache with an e-mail reader that is based upon the email-sync [7] program that reads files from an IMAP server and caches them on local disk. We modified the program to use our cache to adaptively decide whether to read cached mail from disk or from the network. To generate a representative workload, we instrumented our mail reader (exmh) to record the time, file name, and size of any mail that we sent or received. We recorded activity over a one week period and then selected a 15 minute period of heavy e-mail activity to replay. The selected segment corresponds to a user coming in at the beginning of the day and reading accumulated mail — there are 41 messages read in the trace.

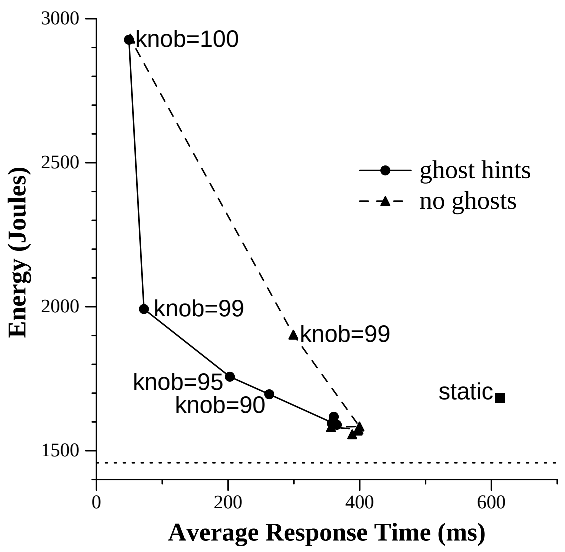
When replaying the trace, we assume that the handheld has already contacted the IMAP server and cached mail locally to provide disconnected access. Thus, the cache may choose to read e-mail from disk or from the server. We replay the trace, preserving the interactive think-time between each event. When a trace event shows that the user has sent an e-mail, we send a message of the same size to the server.

For Web experiments, we used the Dillo Web browser [6]. Since Dillo does not natively support a disk cache, we used the wwwoffle disk caching proxy [3] to access local storage. Unlike, the e-mail client, we cannot reasonably expect the disk cache to contain all data that we access during the trace. Since recent studies [29] have shown that the maximum hit rate of a Web cache is approximately 50–60%, we only cache half of the files that we will access on disk before beginning the trace. The subset that is omitted is chosen randomly, but is the same for all experiments. For uncached files, the Web browser must contact the server to retrieve the data; it then writes the data to the disk cache. For locally cached files, our adaptive cache may choose to fetch the files off disk or off the network. We selected a 20 minute trace from the set of 1996 Berkley Web traces [11]. This trace accesses 116 unique Web objects. We chose the client trace that had file size distribution closest to average and used the first 20 minutes of the trace. When replaying the trace, we preserve the recorded inter-arrival time of Web requests.

5.2 Effectiveness of energy-aware caching

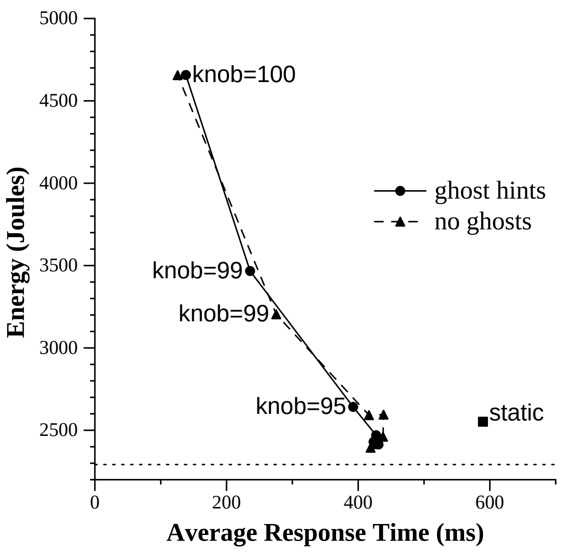
We first compared our adaptive cache to the existing static disk cache implementation for each application. For the static case, the application always reads data from disk if it is cached locally. The static disk experiments use the default device power management mechanisms; e.g. PSM (default 802.11b power management) for the network interface, and the microdrive's ABLE adaptive power manager. For our adaptive cache, we use STPM — the global knob is set to 50 to specify that energy conservation and performance are to be given equal priority.

As Figure 7 shows, our adaptive cache delivers substantial performance improvement and some energy savings compared to the default caching strategy. The left graph shows the average interactive response time for each application — this represents the average time needed to fetch a file. The right graph shows the total amount of energy used by the handheld during trace execution. For the e-mail application, our power management strategy reduces average response time by 42% and total energy usage by 5%. For the Web application, our strategy reduces average response



This graph shows average response time and total energy used by the e-mail reader for seven different STPM knob values. The solid line shows results with ghost hints; the dashed line shows results without. Knob values lower than those labeled yield roughly similar results. The dotted line on the bottom graph is a lower bound on the minimum energy needed to execute the trace.

Figure 10: Effect of ghost hints for e-mail



This graph shows average response time and total energy used by the Web browser for seven different STPM knob values. The solid line shows results with ghost hints; the dashed line shows results without. Knob values lower than those labeled yield roughly similar results. The dotted line on the bottom graph is a lower bound on the minimum energy needed to execute the trace.

Figure 11: Effect of ghost hints for Web

time by 27% and total energy usage by 9%. One reason why the relative benefit of our caching strategy is lower for the Web trace is that several of the accesses hit in the Web browser's memory cache and thus receive no benefit.

5.3 Adjusting the knob

We next examined how changing the relative priority of performance and energy conservation affects the behavior of our system. We specified different knob values between 0 (maximum energy conservation) and 100 (maximum performance). Figure 8 shows results for the e-mail reader. The solid line shows the behavior of our system for different knob values; the box to the far right shows the static disk strategy as a comparison point. The dashed line at the bottom of the graph shows a loose lower bound on the minimum energy usage that could possibly be achieved while running this experiment. We calculated this value by multiplying the power consumed by the iPAQ when idle (with disk in standby and network in PSM) by the total time taken to execute the trace when no power management is employed. Thus, this lower bound shows the energy that would be needed if disk and network accesses could be performed at no energy cost.

Changing the knob value allows the user to improve performance at the cost of increased energy usage. For the e-mail reader, there appears to be a knee in the curve. Knob values between 0 and 70 yield results similar to the default value of 50. At a knob value of 95, interactive response time is reduced by 67% compared to the static disk cache, while energy usage is increased by only 4%. At a knob value of 99, interactive response time is reduced by 89% compared to the static cache, while energy usage is increased 18%. At a knob value of 100, no power management is employed.

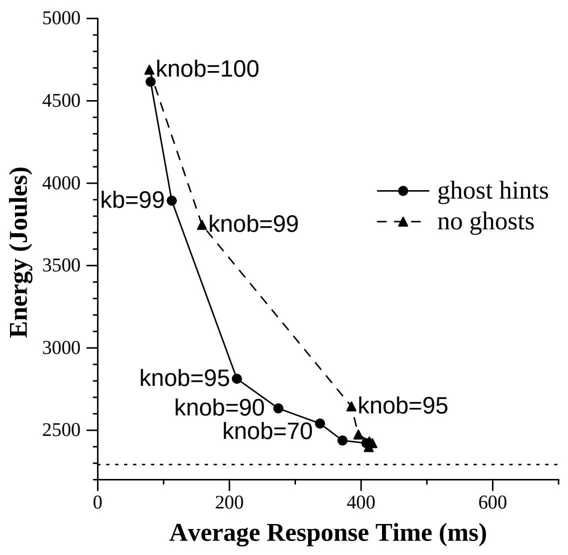
For the Web application in Figure 9, tuning the knob has less effect. In addition, the curve does not exhibit much of a knee. Knob values of 0–90 have similar behavior. Compared to the default value of 50, using a knob value of 95 reduces interactive response time by an additional 9% at an energy cost of 7%. Using a knob value of 99 reduces response time by 45% at an energy cost of 44%.

One reason why marginal changes in the knob value create little difference in system behavior for low knob values is that there is little room for further energy improvement. At the default knob of 50, our lower bound calculation tells us that the Web application could decrease its energy usage by no more than 9% and the e-mail application could reduce energy usage by no more than 5%.

From these results, we conclude that the knob value gives the user the potential to enact substantial tradeoffs between performance and energy conservation. Further, increasing the knob value never substantially increases energy usage and decreasing the knob value never substantially hurts performance. However, we are disappointed that the impact of this control seems application-dependent. Our next step is therefore to implement feedback control that automatically adjusts the knob value to meet specific energy and response-time goals. For example, we could use feedback to dynamically adjust the knob to maximize energy conservation while ensuring that average interactive response time is less than 200 ms. Alternatively, we could meet specific battery-lifetime goals as is done in Odyssey [10].

5.4 The importance of ghost hints

We next examine the importance of issuing ghost hints. For this purpose, we built a version of our cache manager that does not issue ghost hints; all other elements of our architecture remain the same. The dashed line in Figure 10 shows results without ghost hints for the e-mail application. The solid line shows results with ghost hints enabled. For



This graph shows average response time and total energy used by the Web browser for the case when all accesses hit in the disk cache. Knob values lower than those labeled yield roughly similar results. The dotted line on the bottom graph is a lower bound on the minimum energy needed to execute the trace.

Figure 12: Effect of ghost hints for full Web cache

this application, it is clear that ghost hints yield a substantial benefit, especially when performance is a high priority. Without ghost hints, a knob value of 99 yields results clearly inferior to those that can be achieved with ghost hints (several points on the solid line yield better energy conservation and performance). For small knob values, the effect is less clear, yet the version with ghost hints appears to slightly outperform the version without ghost hints.

For the Web application in Figure 11, ghost hints appear to have no real effect on system behavior. In our examination of the results, we found that the STPM disk module was not being aggressive enough in spinning up the disk for low knob values. Because only half of the accesses hit in the cache, we were less likely to see a run of accesses clustered together; thus, ghost hints tended to be issued farther apart in time. Consequently, the STPM disk module rarely spins up the disk.

To validate this theory, we constructed an (admittedly artificial) benchmark in which all Web objects are in the disk cache and reran the trace. The results of this new benchmark are shown in Figure 12. Here, we see that ghost hints do exhibit a substantial positive effect on the system, giving some evidence to support our theory.

From these results, we conclude that ghost hints are a valuable addition to our system. Ghost hints yield substantial benefit for some workloads, and do no harm in the situations where they seem ineffective. We also believe that the break-even heuristic we are currently employing for ghost hints may be insufficiently aggressive. We therefore plan to investigate whether other heuristics will allow ghost hints to yield more consistent improvement across applications.

6. RELATED WORK

To the best of our knowledge, our system is the first to allow applications to efficiently access data on multiple I/O devices by explicitly considering device-specific power management policies. We have also shown that is possible to gracefully manage multiple devices that export custom power management interfaces by using a self-tuning power management layer to translate a global priority for performance and energy conservation into device-specific policies.

Zheng et al. [32] performed a detailed characterization of the energy needed to read and write data to several network and storage devices. Their study helps motivate our work by showing that the cost of accessing many I/O devices varies substantially depending upon power mode.

Weissel's cooperative I/O interface [28] helps coordinate application activity with device power management. Applications specify accesses that can be deferred until the disk is spun up by another access. We could have used this interface to implement the write queue in our cache manager, although an explicit user level queue would still be needed to avoid creating a thread for each cache write. More generally, Papathanasiou and Scott [23, 24] explore how aggregation of disk accesses can improve energy-efficiency, and Heath et al. [12, 13] investigate the use of compiler transformations to produce such aggregation.

ACPI [15] allows applications to query device power modes. Unlike our power manager, ACPI does not disclose performance and energy characteristics. Further, the complexity of the ACPI standard has hindered deployment of ACPI on many operating systems. Our approach, in contrast, attempts to expose a minimal interface to applications.

Shih et al. [25] propose wake-on-wireless, a system which uses a low-power network to signal packet arrivals and guide the power management of a high-power 802.11b network interface. This system can logically be viewed as a single network interface since the low-power network is not intended for data transmission. When multiple wireless networks are available for data transmission, we believe that our system could guide power management decisions for both interfaces.

ECOSystem [30] enforces application-specific energy allocations by descheduling processes that exceed their energy budgets. Their currentcy abstraction [31] extends allocation decisions to I/O devices such as the network and disk. Thus, currentcy can be used to describe allocation policies that span multiple devices. We view allocation as orthogonal to our goals: self-tuning power management tries to achieve the best combination of performance and energy conservation given a specific application workload, but does not limit which requests are serviced by devices.

Odyssey [10] also uses adaptation to reduce energy, but focuses on a different issue. Odyssey applications use quality adaptation to conserve energy. In our system, applications adaptively change when and where they read data in order to save energy and improve performance.

There is a rich body of existing work in both network [17, 18, 26] and disk [8, 9, 20] power management. In contrast to our work, this prior research focuses only on the management of a single device. Additionally, our work shows how power management policies can be improved by considering additional context such as base power and the current priority of performance and energy conservation.

7. CONCLUSIONS AND FUTURE WORK

The goal of self-tuning power management is to provide the mobile user with better performance, longer battery lifetime, and more intuitive control over existing power management mechanisms. We believe that the research reported here is an important step in this direction.

Adaptive caching has proven to be a good choice for the first prototype of our power management interfaces. The read-only nature of the application and our choice to limit adaptation to one network and one storage device have provided a restricted domain in which we could develop our ideas. In the future, we plan to move beyond this domain to less restricted arenas.

One exciting opportunity is distributed file systems. Since clients in this domain may modify as well as read data, the caching infrastructure must provide energy-efficiency for both the read and write paths. One possibility along these lines is to use relaxed cache consistency to delay and aggregate writes to network and storage devices. Further, since many mobile computers now sport several types of storage, including mobile hard drives, flash memory, and USB sticks, we can expand our decision space to include the possibility of reading from and writing to multiple storage devices. One challenge that this suggests is deciding when to issue ghost hints in the presence of many I/O devices: should hints be issued only to the single device that might offer the most ideal performance or energy conservation, or should hints be issued to all devices that could do better than the device currently being used?

Another area of planned research is an investigation into the use of feedback in setting the global tradeoff between performance and energy conservation. We also plan to explore alternatives to the threshold-based strategy that we are using for power management. The end result, we hope, will be better operating system control over power management and a more enjoyable experience for mobile users.

Acknowledgments

We would like to thank our shepherd, Carla Ellis, and the anonymous reviewers for several helpful suggestions that improved the quality of this paper. This research was partially supported by the National Science Foundation under grant CCR-0306251 and by an equipment grant from Intel Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, Intel, the University of Michigan, or the U.S. government.

8. REFERENCES

[1] Anand, M., Nightingale, E. B., and Flinn, J. Self-tuning wireless network power management. In *Proceedings of the 9th Annual Conference on Mobile Computing and Networking (MOBICOM '03)* (San Diego, CA, September 2003), pp. 176–189.

[2] Arpaci-Dusseau, A., and Arpaci-Dusseau, R. Information and control in gray-box systems. In *Proccedings of the 18th ACM Symp. on Operating Systems Principles* (Banff, Canada, October 2001), pp. 43–56.

[3] Bishop, A. M. World wide web offline explorer 2.8. http://www.gedanken.demon.co.uk/wwwoffle/, October 2003.

[4] Carson, M. *Adaptation and Protocol Testing thorugh Network Emulation*. NIST, http://snad.ncsl.nist.gov/itg/nistnet/slides/index.htm.

[5] Cisco Systems, Inc. *Cisco Aironet wireless LAN client adapters installation and configuration guide for Linux*.

[6] Dillo web browser 0.7.2. http://www.dillo.org/, April 2003.

[7] Doron, E. Emailsync - email synchronizer 0.0.8. http://www.savan.com/erez/emailsync.html, July 2002.

[8] Douglis, F., Krishnan, P., and Bershad, B. Adaptive disk spin-down policies for mobile computers. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing* (Ann Arbor, MI, April 1995), pp. 121–137.

[9] Douglis, F., Krishnan, P., and Marsh, B. Thwarting the power-hungry disk. In *Proceedings of 1994 Winter USENIX Conference* (San Francisco, CA, January 1994), pp. 292–306.

[10] Flinn, J., and Satyanarayanan, M. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)* (Kiawah Island, SC, December 1999), pp. 48–63.

[11] Gribble, S. D. UC Berkeley Home IP HTTP Traces. http://www.acm.org/sigcomm/ITA/.

[12] Heath, T., Pinheiro, E., and Bianchini, R. Application-supported device management for energy and performance. In *Proceedings of the 2002 Workshop on Power-Aware Computer Systems* (February 2002), pp. 114–123.

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

[14] Hitachi Global Storage Technologies. *Hitachi Microdrive Hard Disk Drive Specifications*, January 2003.

[15] Intel, Microsoft, and Toshiba. *Advanced Configuration and Power Interface Specification*, February 1998. http://www.acpi.info.

[16] Kistler, J. J., and Satyanarayanan, M. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10*, 1 (February 1992).

[17] Krashinsky, R., and Balakrishnan, H. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM '02)* (Atlanta, GA, July 2002).

[18] Kravets, R., and Krishnan, P. Application-driven power management for mobile communication. *ACM Wireless Networks 6*, 4 (2000), 263–277.

[19] Lampson, B. Hints for computer system design. In *Proceedings of the 9th Symposium on Operating Systems Principles* (Bretton Woods, NH, October 1983), pp. 33–48.

[20] Lu, Y.-H., and De Micheli, G. Adaptive hard disk power management on personal computers. In *Proceedings of the 9th Great Lakes Symposium on VLSI* (Ypsilanti, MI, March 1999), pp. 50–53.

[21] Lu, Y.-H., and Micheli, G. D. Comparing system-level power management policies. *IEEE Design and Test of Computers 18*, 2 (March 2001), 10–19.

[22] Mozilla web browser. http://www.mozilla.org.

[23] Papathanasiou, A. E., and Scott, M. L. Increasing disk burstiness through energy efficiency. Tech. Rep. 792, Computer Science Department, University of Rochester, November 2002.

[24] Papathanasiou, A. E., and Scott, M. L. Energy efficiency through burstiness. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications* (Monterey, CA, October 2003), pp. 444–53.

[25] Shih, E., Bahl, P., and Sinclair, M. J. Wake on wireless: An event-driven energy saving strategy for battery operated devices. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM '02)* (Atlanta, GA, September 2002).

[26] Simunic, T., Benini, L., Glynn, P., and Micheli, G. D. Dynamic power management for portable systems. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM '00)* (Boston, MA, August 2000), pp. 11–19.

[27] USB Flash Store. http://www.usbflashstore.com (as of 11/4/03).

[28] Weissel, A., Beutel, B., and Bellosa, F. Cooperative I/O: A novel I/O semanatics for energy-aware applications. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, December 2002), pp. 117–129.

[29] Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A. R., and Levy, H. M. On the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles* (Kiawah Island, SC, December 1999), pp. 16–31.

[30] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. ECOSystem: Managing energy as a first class operating system resource. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)* (San Jose, CA, October 2002).

[31] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. Currentcy: A unifying abstraction for expressing energy management policies. In *Proceedings of the 2003 USENIX Annual Technical Conference* (San Antonio, TX, June 2003), pp. 43–56.

[32] Zheng, F., Garg, N., Sobti, S., Zhang, C., Joseph, R. E., Krishnamurty, A., and Wang, R. Y. Considering the energy consumption of mobile storage alternatives. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (October 2003).