Managing Battery Lifetime with Energy-Aware Adaptation

JASON FLINN
University of Michigan
and
M. SATYANARAYANAN
Carnegie Mellon University

We demonstrate that a collaborative relationship between the operating system and applications can be used to meet user-specified goals for battery duration. We first describe a novel profiling-based approach for accurately measuring application and system energy consumption. We then show how applications can dynamically modify their behavior to conserve energy. We extend the Linux operating system to yield battery lifetimes of user-specified duration. By monitoring energy supply and demand and by maintaining a history of application energy use, the approach can dynamically balance energy conservation and application quality. Our evaluation shows that this approach can meet goals that extend battery life by as much as 30%.

Categories and Subject Descriptors: D.4.0 [**Operating Systems**]: General

General Terms: Management, Measurement

Additional Key Words and Phrases: Power management, adaptation

1. INTRODUCTION

Energy is a vital resource for mobile computing. The amount of work one can perform while mobile is fundamentally constrained by the limited energy supplied by one's battery. Unfortunately, despite considerable effort to prolong the battery lifetimes of mobile computers, no silver bullet for energy management

This research was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Materiel Command (AFMC) under contracts F19628-93-C-0193 and F19628-96-C-0061, DARPA, the Space and Naval Warfare Systems Center (SPAWAR)/U.S. Navy (USN) under contract N660019928918, the National Science Foundation (NSF) under contracts CCR-9901696 and ANI-0081396, IBM Corporation, Intel Corporation, AT&T, Compaq, Hughes, and Nokia. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either express or implied, of DARPA, AFMC, SPAWAR, USN, the NSF, IBM, Intel, AT&T, Compaq, Hughes, Nokia, Carnegie Mellon University, the University of Michigan, the U.S. Government, or any other entity.
Authors' addresses: J. Flinn, University of Michigan, EECS 2221, 1301 Beal Ave., Ann Arbor, MI 48109-2122; email: jflinn@umich.edu; M. Satyanarayanan, Computer Science Deptartment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213; email: satya@cs.cmu.edu.

has yet been found. Instead, there is growing consensus that a comprehensive approach is needed—one that addresses *all* levels of the system: circuit design, hardware devices, the operating system, and applications.

In this article, we show that *energy-aware adaptation*, the dynamic balancing of energy conservation and application quality, is an essential part of a comprehensive energy management solution. Occasionally, energy usage can be reduced without affecting the perceived quality of the system. More often, however, significant energy reduction perceptibly impacts system behavior. The effective design of mobile software thus requires striking the appropriate balance between application quality and energy conservation.

It is incorrect to make static decisions that arbitrate between these two competing goals. Dynamic variation in time operating on battery power, hardware power requirements, application mix, and user specifications all affect the balance between quality and energy conservation. Energy-aware adaptation surmounts these difficulties by making decisions dynamically. Applications statically specify *possible* tradeoffs between energy and quality, but defer decisions about which tradeoffs to make until execution. At that time, the system uses additional information, including energy supply and demand, to advise applications which tradeoffs to make.

We begin in the next section by developing a tool, called PowerScope, that measures energy usage and attributes it to application components. This tool forms the basis for our measurement methodology—it also is a useful artifact that enables the development of energy-efficient code. In Section 3, we use PowerScope to study how applications can reduce the energy usage of the computers on which they execute by varying their quality levels. Then, in Section 4, we use the results of this study to design and implement operating system support for such energy-aware applications. Our implementation monitors energy supply and demand and uses feedback to advise applications how to balance quality and energy conservation. We conclude with a discussion of related work and a summary of results.

This article builds upon our previously reported work in energy management [Flinn and Satyanarayanan 1999a] by providing a detailed description and evaluation of the PowerScope energy profiler, as well as a more complete study of energy-aware applications. Further, we show how operating system support can be improved by observing applications as they execute, learning functions that predict their future energy usage, and using this information to directly adjust application fidelity.

2. PROFILING APPLICATION ENERGY USAGE

There are currently few tools that allow software developers to accurately measure how their systems and applications impact battery lifetime. Low-level approaches such as instruction-level profiling [Tiwari et al. 1994, 1996] and hardware simulation [Brooks et al. 2000; Vijaykrishnan et al. 2000] do not scale to multithreaded applications with tens of thousands of lines of source code. High-level approaches such as direct measurement of battery drain through APM [Intel Corporation and Microsoft Corporation 1996]

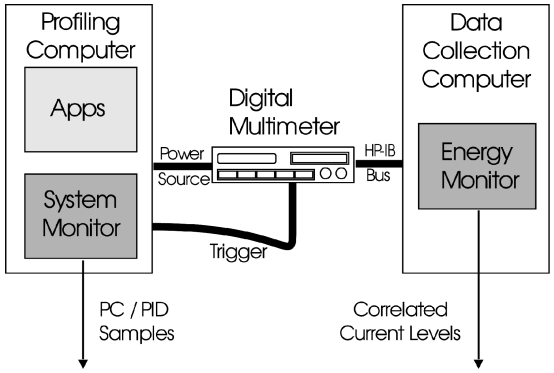


Fig. 1. PowerScope architecture.

or ACPI [Intel, Microsoft, and Toshiba 1998] interfaces provide insufficient detail.

We have built an energy profiling tool, called PowerScope [Flinn and Satyanarayanan 1999b], that fills this gap. Using PowerScope, one can determine what fraction of the total energy consumed during a certain time period is due to specific processes in the system. Further, one can drill down and determine the energy consumption of different procedures within a process.

The design of PowerScope follows from its primary purpose: enabling application developers to build energy-efficient software. PowerScope's design scales to complex applications, which may consist of several concurrently executing threads of control, and which may run on a variety of mobile platforms. For both simple and complex applications, PowerScope provides developers detailed and accurate information about energy usage.

2.1 Implementation

2.1.1 *Overview.* As shown in Figure 1, PowerScope uses statistical sampling to profile the energy usage of a computer system. To reduce overhead, profiles are generated by a two-stage process. During the data collection stage, the tool samples both the power consumption and the system activity of the profiling computer. PowerScope then generates an energy profile from this data during a later analysis stage. Because the analysis is performed off-line, it creates no profiling overhead.

During data collection, PowerScope uses two computers: a profiling computer, on which applications execute, and a data collection computer that reduces overhead. A digital multimeter samples the power consumption of the profiling computer. We require that this multimeter have an external trigger input and output, as well as the ability to sample DC current or voltage at high frequency. The present implementation uses a Hewlett Packard 3458a digital multimeter, which satisfies both requirements. The data collection computer controls the multimeter and stores current samples.

An alternative implementation would be to perform measurement and data collection entirely on the profiling computer, perhaps using an on-board digital

multimeter. However, this implementation makes it very difficult to differentiate the energy consumed by the profiled applications from the energy used by data collection. Further, our present implementation makes switching the measurement equipment to profile different hardware platforms much easier.

The functionality of PowerScope is divided among three software components. Two components, the System Monitor and Energy Monitor, share responsibility for data collection. The System Monitor samples system activity on the profiling computer by periodically recording information that includes the program counter and process identifier of the currently executing process. The Energy Monitor runs on the data collection computer, and is responsible for collecting and storing current samples. Because data collection is distributed across two monitor processes, PowerScope synchronizes the collection by having the multimeter generate an external trigger signal after taking a measurement. The signal causes an interrupt on the profiling computer, during which the System Monitor samples system activity.

The final software component, the Energy Analyzer, uses the raw sample data collected by the monitors to generate the energy profile. The analyzer runs on the profiling computer since it uses the symbol tables of executables and shared libraries to map samples to specific procedures. There is an implicit assumption in this method that the executables being profiled are not modified between the start of profile collection and the running of the off-line analysis tool.

2.1.2 *The System Monitor.* The System Monitor consists of a device driver which collects sample data and a user-level daemon process that reads the samples from the driver and writes them to a file. Since the device driver is a Linux loadable kernel module, PowerScope runs without modification to kernel source code.

The design of the System Monitor is similar to the sampling components of Morph [Zhang et al. 1997] and DCPI [Anderson et al. 1997]. It samples system activity when triggered by the digital multimeter. Each 12-byte sample records the value of the program counter (PC) and the process identifier (PID) of the currently executing process, as well as additional information such as whether the system is currently handling an interrupt. This assumes that the profiling computer is a uniprocessor—a reasonable assumption for a mobile computer.

Samples are written to an in-kernel circular buffer. This buffer is emptied by the user-level daemon, which writes samples to a file. The daemon is triggered when the buffer grows more than seven-eighths full, or by the end of data collection.

The System Monitor records a small amount of additional information that is used to generate profiles. First, it associates each currently executing process with the pathname of an executable. Then, for each executable it records the memory location of each loaded shared library and associates the library with a pathname. The information is written to the sample buffer during data collection, and is used during off-line analysis to associate each sample with a specific executable image.

The API in Figure 2 allows applications to control profiling. The user-level daemon calls `pscope_init` to set the size of the kernel sample buffer. Since there

```
pscope_init (u_int size);
pscope_read (void* sample, u_int size, u_int* ret_size);
pscope_start (void);
pscope_stop (void);
```

Fig. 2. PowerScope API.

is a tension between excessive memory usage and frequent reading of the buffer by the daemon, the buffer size has been left flexible to allow efficient profiling of different workloads. The daemon calls `pscope_read` to read samples out of the buffer. The `pscope_start` and `pscope_stop` system calls allow application programs to precisely indicate the period of sample collection. Multiple sets of samples may be collected one after the other; each sample set is delineated by start and end markers written into the sample buffer.

2.1.3 *The Energy Monitor.* The Energy Monitor runs on the data collection computer. It configures the multimeter to periodically sample the power usage of the profiling computer. The specific method of power measurement depends upon the system being profiled. For many laptop computers, the simplest method is to sample the current drawn through the laptop's external power source. Usually, the voltage variation is extremely small; for example, it is less than 0.25% for the IBM 701C and 560X laptops. Therefore, current samples alone are sufficient to determine the energy usage of the system. The battery is removed from the laptop while measurements are taken to avoid extraneous power drain caused by charging. Current samples are transmitted asynchronously to the Energy Monitor, which stores them in a file for later analysis.

We use an alternative method for systems such as the Compaq Itsy v1.5 pocket computer that provide internal precision resistors for power measurement [Viredaz 1998]. The Energy Monitor configures the multimeter to measure the instantaneous differential voltage, V_{diff}, across a 20 m resistor located in the main power circuit. The instantaneous current, I, can therefore be calculated as $I = V_{diff}/0.02$. Since the voltage supplied by the external power supply, V_{supp}, does not vary significantly, these measurements are sufficient to calculate instantaneous power usage, P, as $P = V_{supp} * I$. Further, because the Itsy contains additional resistors, the same method can profile the isolated power usage of Itsy subsystems. If the profiling computer is battery-powered, concurrent sampling of voltage [Farkas et al. 2000] is needed.

Sample collection is driven by the multimeter clock. Synchronization with the System Monitor is provided by connecting the multimeter's external trigger input and output to I/O pins on the profiling computer. Immediately after the multimeter takes a power sample, it toggles the value of an input pin. This causes a system interrupt on the profiling computer, during which the System Monitor samples system activity. Upon completion, the System Monitor triggers the next sample by toggling an output pin (unless profiling has been halted by `pscope_stop`). The multimeter buffers this trigger until the time to take the next

sample arrives. This method ensures that the power samples reflect application activity, rather than the activity of the System Monitor.

Originally, PowerScope used the clock of the profiling computer to drive sample collection. Although simpler to implement, that design had the disadvantage of biasing the profile values of activities correlated with the system clock. We have since modified PowerScope to drive sample collection from the multimeter. The lack of synchronization between the multimeter and profiling computer clocks introduces a natural jitter that makes clock-related bias very unlikely. Using the multimeter clock also allows PowerScope to generate interrupts at a finer granularity than that allowed by using kernel clock interrupts. The user may specify the sample frequency as a parameter when the Energy Monitor is started. The maximum sample frequency for our current multimeter is approximately 700 samples/s.

2.1.4 *The Energy Analyzer*. The Energy Analyzer generates an energy profile of system activity. Total energy usage can be calculated by integrating the product of the instantaneous current and voltage over time. One can approximate this value by simultaneously sampling both current and voltage at regular intervals of time Δt. Further, because we measure externally powered systems, voltage is constant within the limits of accuracy for which we are striving. PowerScope therefore calculates total energy over n samples using a single measured voltage value, V_{meas}, as follows:

$$E \approx V_{meas} \sum_{t=0}^{n} I_t \Delta t. \tag{1}$$

The Energy Analyzer associates each current sample collected by the Energy Monitor with the corresponding sample collected by the System Monitor. It assigns each sample to a process bucket using the recorded PID value. Samples that occurred during the handling of an asynchronous interrupt, such as the receipt of a network packet, are not attributed to the currently executing process but are instead attributed to a bucket specific to the interrupt handler. If no process was executing when the sample was taken, the sample is attributed to a kernel bucket. The energy usage of each process is calculated as in Equation (1) by summing the current samples in each bucket and multiplying by the measured voltage (V_{meas}) and the sample interval (Δt).

The Energy Analyzer then generates a summary of energy usage by process, such as the one shown in Figure 3(a). Each entry displays the total time spent executing the process, the total energy usage, and the average power usage. It then repeats the above steps for each process to determine energy usage by procedure. The process and shared library information stored by the System Monitor is used to reconstruct the memory address of each procedure from the symbol tables of executables and shared libraries. Then, the PC value of each sample is used to place the sample in a procedure bucket. When the profile is generated, kernel procedures and procedures that reside in shared libraries are displayed separately. Figure 3(b) shows a partial profile.

A drawback of this approach is that a process using the CPU can be incorrectly assigned the energy cost of asynchronous I/O activity such as disk reads

Process	Elapsed Time (s)	Total Energy (J)	Average Power (W)
/obj/odyssey/bin/janus	40.521	489.522	12.081
kernel	40.572	301.210	7.424
Interrupts-Wavelan	27.654	296.287	10.714
/obj/odyssey/bin/xanim	18.073	218.458	12.087
/usr/X11R6/bin/XF86_SVGA	13.369	162.659	12.167
/obj/odyssey/bin/viceroy	11.730	141.101	12.029
/obj/odyssey/bin/editor	2.130	25.087	11.776
/usr/bin/netscape3	1.495	17.791	11.901

(a) Partial summary of energy usage by process

Energy Usage Detail for process /obj/odyssey/bin/viceroy

Procedure	Elapsed Time (s)	Total Energy (J)	Average Power (W)
Internal_Signal	0.210	2.585	12.327
ExaminePacket	0.165	1.939	11.763
Dispatcher	0.160	1.872	11.693
sftp_DataArrived	0.106	1.285	12.162
IOMGR_CheckDescriptors	0.096	1.159	12.064
IOMGR_Select	0.078	0.955	12.177

(b) Partial detail of process energy usage

Fig. 3. Sample energy profile.

initiated by another process. One solution we have considered, but not implemented, is to simultaneously sample the power usage of the disk, record the disk requests issued by each process, and separately assign disk energy costs to each process. A similar approach could also allocate the energy cost of network activity.

2.2 Validation

For PowerScope to be effective, it must accurately determine the energy cost of processes and procedures. Further, it must operate with a minimum of overhead on the system being measured to avoid significantly perturbing the profile results.

We created benchmarks to assess how successful PowerScope is in meeting both of these goals. We ran each benchmark on two hardware platforms, the Compaq Itsy v1.5 pocket computer [Bartlett et al. 2000] and the IBM ThinkPad 560X laptop.

2.2.1 *Accuracy.* There are several factors that potentially limit PowerScope's accuracy. First, the digital multimeter's power measurements are not truly instantaneous; the multimeter's analog/digital converter must measure the input signal over a period of time. However, this period, or *integration time*, is normally quite small. In the case of the HP 3458a multimeter used for these experiments, the minimum integration time is only 1.4 s. Second, there will be some capacitance in the computer system being measured. High-frequency changes in power usage may not be measurable at the point in the circuit where

the multimeter probes are attached. Finally, there is a delay between the time when the multimeter takes a measurement and the time when the corresponding kernel sample is taken; this delay includes time to propagate an electrical signal to the profiling computer and time to handle the corresponding hardware interrupt. If a process or procedure is of sufficiently short duration, a sample taken during its execution may be incorrectly attributed to a process or procedure that executes later. Combined, these factors limit PowerScope's accuracy; that is, there will be some minimum event duration below which PowerScope is unable to accurately determine an event's power usage.

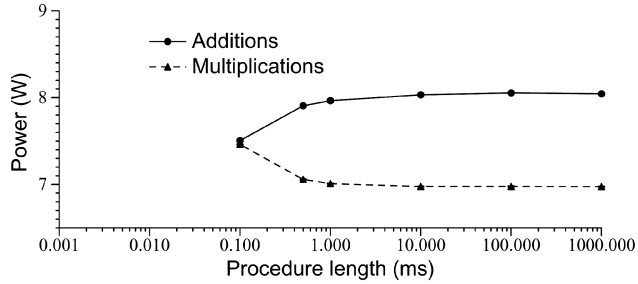
We measured the minimum event duration by running a benchmark which alternates execution between two different procedures. Each procedure has a known power usage and runs for a configurable length of time. When these procedures are of sufficiently long duration, for example, 1 s, PowerScope can accurately determine the power usage of each procedure. However, as the duration of the two procedures is shortened, PowerScope will eventually be unable to successfully determine their individual power usages. To ensure maximum accuracy, we used the highest sampling rate supported by the multimeter for these measurements—approximately 700 samples/s. Since this benchmark is CPU-bound, all activity is synchronous. If the benchmark included asynchronous I/O, PowerScope's accuracy would decrease since it does not attribute such activity to the initiating process.

Figure 4(a) shows the results of running the benchmark on the 560X laptop. The first procedure performs additions in an unrolled loop and has a power usage of 8.04 W (measured with a duration of 1 s). The second procedure performs multiplications in an unrolled loop and has a power usage of 6.97 W. While it may seem unintuitive that multiplication requires less power than addition, the multiplication procedure executes less instructions per unit of time than the addition procedure. Because a multiplication instruction takes more than one cycle to execute, the total *energy* needed to perform a multiplication is higher—effectively, that energy is spread out across multiple cycles.

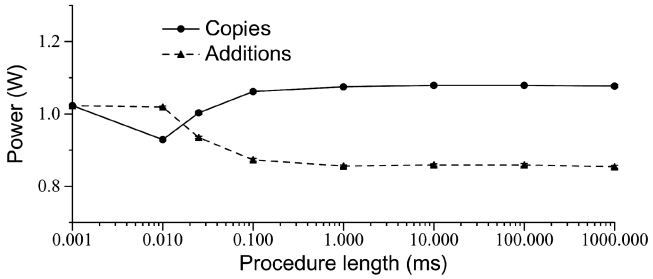
PowerScope correctly reports the individual power usage of each procedure within experimental error for durations of 10 ms. When the procedure length is set to 1 ms, PowerScope reports slightly inaccurate results (within 1% of the correct value). As the procedure duration is further decreased, PowerScope's accuracy also decreases. At a duration of 100 s, PowerScope is unable to distinguish the power usage of individual procedures. In this case, the limiting factor is probably the capacitance of the laptop.

Figure 4(b) shows the results of running the benchmark on the Itsy v1.5. Because the power used to perform multiplications on the Itsy is very similar to the power used to perform additions, we replace the multiplication procedure with one that copies data from one memory location to another. The copies are performed in an unrolled loop and all memory references hit in the first-level data cache.

On the Itsy, PowerScope correctly reports individual power usage within experimental error for durations of 1 ms. The reported values are slightly inaccurate with a procedure length of 100 s (within 3% of the correct value). Interestingly, at 10 s, PowerScope reports a higher power usage for the addition



(a) PowerScope accuracy for ThinkPad 560X



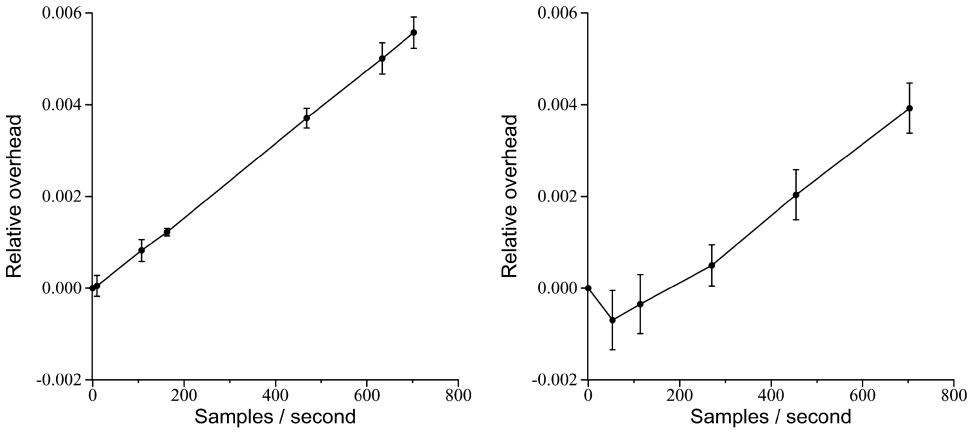
(b) PowerScope accuracy for Itsy v1.5

Fig. 4. PowerScope accuracy. PowerScope's accuracy is shown as a function of the length of the event being measured. Each graph shows the power usage reported for two different procedures which execute alternately. As the procedure length is reduced, PowerScope is eventually unable to distinguish the individual power usage of the two procedures. Each point represents the mean of 10 trials—the (barely noticeable) error bars in each graph show 90% confidence intervals. Note that procedure length, on the x-axis, is displayed using a log scale.

procedure than for the copy procedure. Because these results are significant within experimental error, they strongly indicate that the power measurements are being perturbed by the latency between the time when measurements are taken and the time when the System Monitor samples system activity on the profiling computer. Power samples that should be attributed to one procedure are instead being incorrectly attributed to the other. Note that this phenomenon did not occur with the laptop for any duration that we tried.

2.2.2 *Overhead.* Running PowerScope imposes a small overhead on the system being profiled due to the activity of the System Monitor. The impact can be expressed in terms of both CPU usage and additional energy consumption.

To determine PowerScope's CPU overhead, we measured the execution time of the benchmark described in Section 2.2.1 for a variety of sampling rates, and compared the results to the execution time of the benchmark when PowerScope was not running. For the 560X laptop, we measured latency using the Pentium cycle counter; for the Itsy, we used the Linux `gettimeofday` system call. This benchmark does not include the cost of periodically writing data to a file for long-running profiles. However, because the file write is amortized across a large number of samples, the additional CPU cost is quite low. The energy cost



(a) CPU overhead for ThinkPad 560X

(b) CPU overhead for Itsy v1.5

Fig. 5. PowerScope CPU overhead. PowerScope's relative CPU overhead is shown as a function of the sampling frequency. Each point represents the mean of 10 trials—the error bars in each graph show 90% confidence intervals.

is greater when samples are written to a hard disk; adding asynchronous I/O support to PowerScope could potentially enable the tool to separate out the energy overhead of such disk writes.

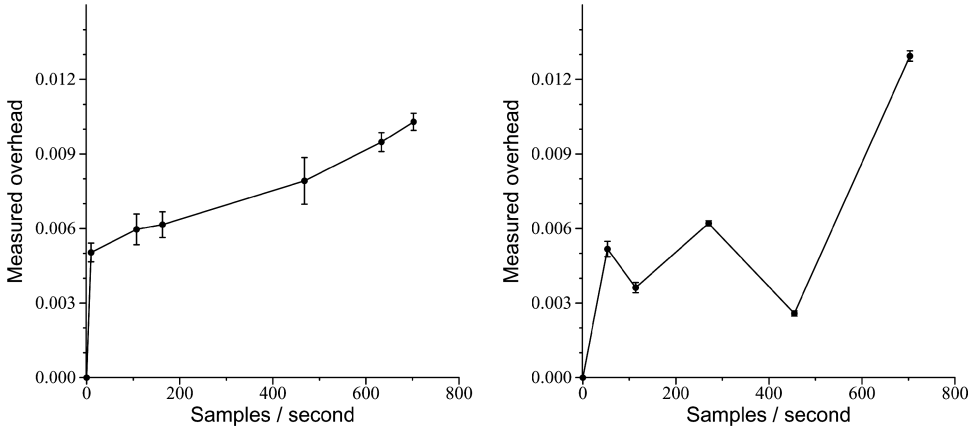
Figure 5 shows the results of these experiments for the two platforms. In both cases, PowerScope's CPU overhead is less than 0.6% at the maximum sampling rate of the multimeter. Note that although two measurements in Figure 5(b) show a negative overhead, the upper bound of each measurement's 90% confidence interval is greater than zero.

To determine PowerScope's energy overhead, we used PowerScope to measure the energy usage of the benchmark described in Section 2.2.1 for a variety of sampling rates. For comparison, we directly measured the energy consumption of the benchmark when PowerScope was not running. As Figure 6 shows, PowerScope's energy overhead is low—about 1.0% for the ThinkPad 560X and 1.3% for the Itsy at the maximum sampling rate. Like the CPU benchmark, this value does not include the cost of periodically writing data to a file for long-running profiles.

While the laptop results show that energy overhead increases fairly regularly with the sample rate, the Itsy results are decidedly more irregular. While it is unclear precisely what leads to this effect, it is possible that different sampling frequencies induce slightly different cache effects when PowerScope writes samples to the in-memory kernel buffer. Such cache effects would be much more noticeable on the Itsy since its cache is smaller and memory usage is a much larger percentage of its overall power budget [Farkas et al. 2000].

3. ENERGY-AWARE ADAPTATION

PowerScope has enabled us to explore how software design choices impact system energy consumption. In this section, we report the results of a detailed



(a) Energy overhead for ThinkPad 560X

(b) Energy overhead for Itsy v1.5

Fig. 6. PowerScope energy overhead. PowerScope's relative energy overhead is shown as a function of the sampling frequency. Each point represents the mean of 10 trials—the error bars in each graph show 90% confidence intervals.

study of application energy usage [Flinn and Satyanarayanan 1999a] and discuss the implications for energy-aware application and operating system design.

Our primary goal was to measure how changes in *fidelity* affect application energy use. Fidelity is an application-specific metric of quality. For example, dimensions of fidelity for a video player are display size and the amount of lossy compression applied to a video stream; for a map viewing application, a dimension of fidelity is the amount of geographic features included on a map. For energy-aware adaptation to be viable, it is crucial that reductions in fidelity lead to energy savings that are both significant and predictable.

We were also keen to confirm that the energy savings from lowering fidelity enhance those achievable through current hardware power management techniques such as spinning down the disk and disabling wireless network receivers. Although these distinct approaches to energy savings seem composable, we wanted to verify this experimentally.

3.1 Methodology

We measured the energy used by four applications: a video player, a speech recognizer, a map viewer, and a Web browser. We first observed the applications as they operated in isolation, and then as they operated concurrently.

We first measured the baseline energy usage for each object at highest fidelity with hardware power management disabled. We next measured energy usage with hardware power management enabled. Then, we successively lowered the fidelity of the application, measuring energy usage at each fidelity with hardware power management enabled. This sequence of measurements is directly reflected in the format of the graphs presenting the results: Figures 7, 9, 11, 12, and 15. Since a considerable amount of data is condensed into these

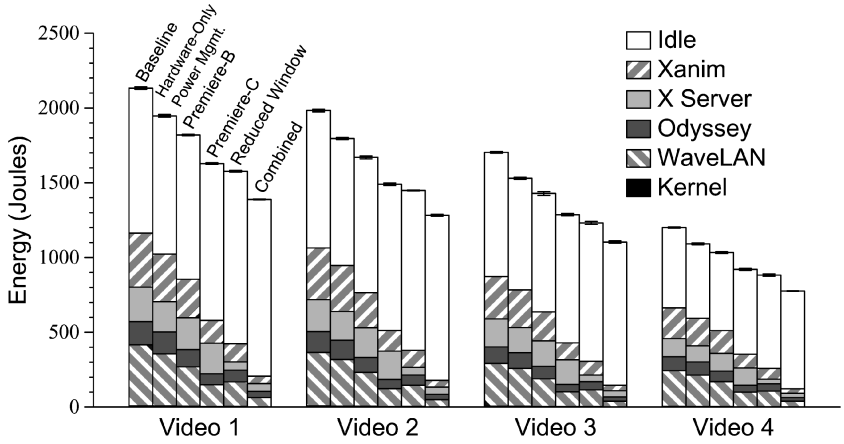


Fig. 7. Energy impact of fidelity for video playing. This shows the total energy used to display four QuickTime/Cinepak videos from 127 to 226 s in length, ordered from right to left above. Each value is the mean of five trials—the error bars show 90% confidence intervals.

graphs, we explain their format here even though their individual contents will not be meaningful until the detailed discussions in Sections 3.3 through 3.6.

For example, consider Figure 7. There are six bars in each of the four data sets on the x-axis; each data set corresponds to a different video clip. The height of each bar shows the total amount of energy consumed by the computer while playing the video clip, and the shadings within each bar apportion system energy usage to each software component. The component labeled "Idle" aggregates samples that occurred while executing the kernel idle procedure—effectively a Pentium `hlt` instruction. The component labeled "WaveLAN" aggregates samples that occurred during wireless network interrupts.

For each data set, the first and second bars, labeled "Baseline" and "Hardware-Only Power Mgmt.," show energy usage at full fidelity with and without hardware power management. The difference between the first two bars gives the application-specific effectiveness of hardware power management. Each of the remaining bars shows the energy usage at a different, reduced fidelity level with hardware power management enabled. The difference between one of these bars and the first bar ("Baseline") gives the combined benefit of hardware power management and fidelity reduction. The difference between one of these bars and the second one ("Hardware-Only Power Mgmt.") gives the additional benefit achieved by fidelity reduction above and beyond the benefit achieved by hardware power management.

The measurements for all bars except "Baseline" were obtained with power management enabled. After 10 s of inactivity, we transitioned the disk to standby mode. Further, we placed the wireless network interface in standby mode except during remote procedure calls or bulk transfers. Finally, we turned off the display during the speech application. The laptop processor used in the study did not support dynamic voltage scaling; the effectiveness of hardware power management would increase if this capability were present.

3.2 Experimental Setup

For this study, the primary client platform was an IBM 560X laptop with 233-MHz Pentium processor, 64 MB of memory, and 2-Mb/s 802.11 wireless network interface. All servers were 200-MHz Pentium Pro desktop computers with 64 MB of memory. The client ran the Linux 2.2 operating system, as well as the Odyssey platform for mobile computing [Noble et al. 1997]. Odyssey provides both an interface for modifying application fidelity and a RPC-based mechanism for accessing remote data. We measured energy use with PowerScope, taking approximately 600 samples/s.

3.3 Video Player

3.3.1 *Description.* We first measured the impact of fidelity on an Xanim-based video player. Xanim fetches video data from a server through Odyssey and displays it on the client. It supports two dimensions of fidelity: varying the amount of lossy compression used to encode a video clip and varying the size of the window in which it is displayed. There are multiple tracks of each video clip on the server, each generated off-line from the full fidelity video clip using Adobe Premiere. They are identical to the original except for size and the level of lossy compression used in frame encoding.

3.3.2 *Results.* Figure 7 shows the total energy used by the laptop to display four videos at different fidelities. At the baseline fidelity, much energy is consumed while the processor is idle because of the limited bandwidth of the wireless network—not enough video data is transmitted to saturate the processor.

For the four video clips, hardware-only power management reduces energy consumption by a mere 9–10%. There is little opportunity to place the network in standby mode since it is nearly saturated. Most of the reduction is due to disk power management—the client's disk remains in standby mode for the entire duration of an experiment.

The bars labeled "Premiere-B" and "Premiere-C" in Figure 7 show the impact of lossy compression. Whereas the baseline video is encoded at QuickTime/Cinepak quality level 2, Premiere-B and Premiere-C are encoded at quality levels 1 and 0, respectively. Premiere-C consumes 16–17% less energy than hardware-only power management. With a higher-bandwidth wireless network, we could fully utilize the processor in the baseline case—the relative energy savings of Premiere-C would then be higher. Unfortunately, the data rate required by the next fidelity higher than baseline, for example, QuickTime/Cinepak quality level 3, is greater than the 2-Mb/s capacity of our wireless network interface.

By examining the shadings of each bar in Figure 7, it can be seen that compression significantly reduces the energy used by Xanim, Odyssey, and the WaveLAN device driver. However, the energy used by the X server is almost completely unaffected by compression. We conjectured that this is because video frames are decoded before they are given to the X server, and the size of this decoded data is independent of the level of lossy compression.

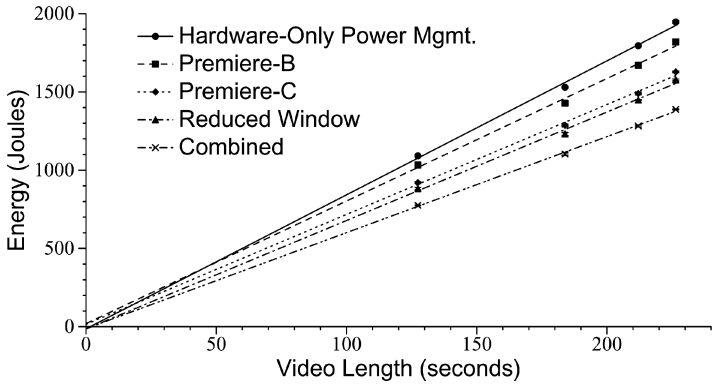


Fig. 8. Predicting video player energy use. Each line is the best linear fit for four videos played at the same fidelity—the (barely noticeable) error bars are 90% confidence intervals.

To validate this conjecture, we measured the effect of halving both the height and width of the display window. As Figure 7 shows, shrinking the window size reduces energy consumption 19–20% beyond hardware-only power management. The shadings on the bars confirm that reducing window size significantly decreases X server energy usage. In fact, within the bounds of experimental error, X server energy consumption is proportional to window area.

Finally, we examined the effect of combining Premiere-C encoding with a display window of half the baseline height and width. This results in a 28–30% reduction in energy usage relative to hardware-only power management. Relative to baseline, using all of the above techniques together yields about a 35% reduction.

From the viewpoint of further energy reduction, the rightmost bar of each data set in Figure 7 seems to offer a pessimistic message: there is little to be gained by further efforts to reduce fidelity. Virtually all energy usage at this fidelity level occurs when the processor is idle. Fortunately, this is precisely where advances in hardware power management can be of the most help. For example, processors such as the TransMeta Crusoe [Klaiber 2000] reduce clock frequency to save power and energy. As the fidelity of the video is reduced, processor speed can also be reduced since the needed computation per unit of time is smaller. Thus, the total energy used by the lowest fidelity will be substantially less.

Figure 8 shows video player energy use as a function of video length for five different levels of fidelity. In each case, hardware power management is enabled. For each fidelity level, four data points show the total energy used to play each of the videos in the study, and the corresponding line shows the best linear fit through these points. From the data, it is clear that the linear model is a good fit—the coefficient of determination (R^2) is greater than 99% for every fidelity. Thus, if one can determine the length of a video, it is possible to accurately predict the energy needed to play it at different fidelities. Of course, it is possible that different encoding schemes may prove to be less predictable.

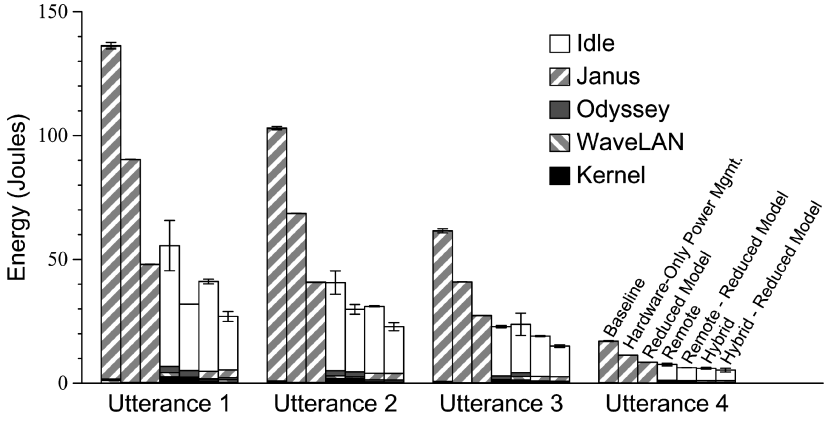


Fig. 9. Energy impact of fidelity for speech recognition. This shows the energy used to recognize four spoken utterances from 1 to 7 s in length, ordered from right to left above. Each measurement is the mean of five trials—the error bars show 90% confidence intervals.

3.4 Speech Recognizer

3.4.1 *Description.* The second application is an adaptive speech recognizer that generates a speech waveform from a spoken utterance and submits it via Odyssey to a local or remote instance of the Janus speech recognition system [Waibel 1996]. Local recognition avoids network transmission and is unavoidable if the client is disconnected. In contrast, remote recognition incurs the delay and energy cost of network communication but can exploit the CPU, memory, and energy resources of a remote server that is likely to be operating from a power outlet rather than a battery. The system also supports a hybrid mode of operation in which the first phase of recognition is performed locally, resulting in a compact intermediate representation that is shipped to the remote server for completion of the recognition. In effect, the hybrid mode uses the first phase of recognition as a type-specific compression technique that yields a factor of 5 reduction in data volume with minimal computational overhead.

Fidelity is lowered by using a reduced vocabulary and a less complex acoustic model. This substantially reduces the memory footprint and processing required, but degrades recognition quality. The system alerts the user of fidelity transitions using a synthesized voice. The use of low fidelity is most compelling in the case of local recognition on a resource-poor disconnected client, although it can also be used in hybrid and remote cases. Although reducing fidelity limits the number of words available, the word-error rate may not increase. Intuitively, this is because the recognizer makes fewer mistakes when choosing from a smaller set of words. This helps counterbalance the effects of reducing the sophistication of the acoustic model.

3.4.2 *Results.* Figure 9 shows laptop energy usage when recognizing four pre-recorded utterances. The baseline measurements correspond to local recognition at high fidelity without hardware power management. Since speech recognition is compute-intensive, almost all the energy in this case is consumed by Janus.

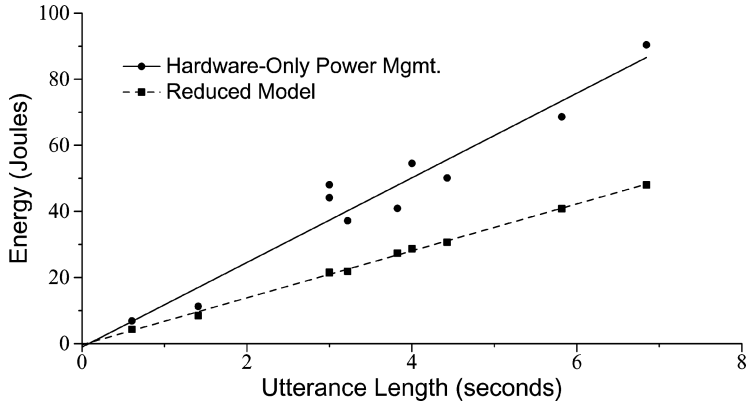


Fig. 10. Predicting speech recognition energy. Each line is the best linear fit for 10 utterances recognized at the same fidelity—the (barely noticeable) error bars are 90% confidence intervals.

Hardware power management reduces client energy usage by 33–34%. Such a substantial reduction is possible because the display can be turned off and both the network and disk can be placed in standby mode for the entire duration of an experiment. This assumes that user interactions occur solely through speech, and that disk accesses can be avoided because the vocabulary, language model, and acoustic model fit entirely in physical memory.

Lowering fidelity by using a reduced speech model results in a 25–46% reduction in energy consumption relative to using hardware power management alone. This corresponds to a 50–65% reduction relative to the baseline.

Remote recognition at full fidelity reduces energy usage by 33–44% relative to using hardware power management alone. If fidelity is also reduced, the corresponding savings are 42–65%. These figures are comparable to the energy savings for remote execution reported for other compute-intensive tasks [Othman and Hailes 1998; Rudenko et al. 1998]. Most of the energy consumed by the client in remote recognition occurs with the processor idle—much of this is time spent waiting for a reply from the server. Lowering fidelity speeds recognition at the server, shortening this interval and yielding client energy savings.

Hybrid recognition offers slightly greater energy savings: 47–55% at full fidelity, and 53–70% at low fidelity. It increases the fraction of energy used by the local Janus system; but this is more than offset by the reduction in network transmission and idle time. Overall, the net effect of combining hardware power management with hybrid, low-fidelity recognition is a 69–80% reduction in energy usage relative to baseline. In practice, the optimal strategy depends on remote resource availability and the user's tolerance for low-fidelity recognition.

Figure 10 examines the predictability of speech recognition energy use. Each data point shows the total energy used to locally recognize one of a set of 10 spoken utterances—this set includes the four utterances from Figure 9. The corresponding lines show the best linear fit through these points. The linear model for the baseline fidelity produces a reasonable fit (93.7% coefficient of determination), while the fit is considerably better for the reduced fidelity (99.8%). While speech recognition is less predictable than video display, these

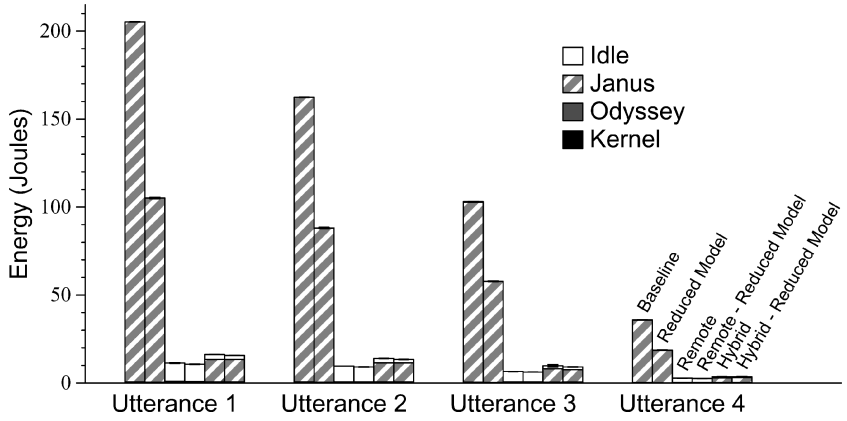


Fig. 11. Energy impact of fidelity for speech recognition on the Itsy v1.5. This shows the energy used to recognize four spoken utterances on an Itsy v1.5 pocket computer—the results can be contrasted with Figure 9 in which the client machine is an IBM ThinkPad 560X laptop. Each measurement is the mean of five trials—the (barely noticeable) error bars show 90% confidence intervals.

results are quite encouraging—simple linear models appear to do a reasonable job of predicting energy use.

3.4.3 *Results for Itsy v1.5.* We explored the impact of different hardware platforms by porting Janus to the Itsy v1.5 pocket computer. Figure 11 shows client energy usage for the same four prerecorded utterances. Since the Itsy lacks a PCMCIA interface, we used a serial link to connect it with the server.

Contrasting Figures 9 and 11 shows that the Itsy consumes more energy than the ThinkPad to perform local speech recognition but significantly less energy to perform hybrid and remote recognition. Local speech recognition runs much slower on the Itsy, partially because its StrongArm processor is less powerful than the ThinkPad's Pentium. Additionally, the Janus recognizer performs a large number of floating-point operations that must be emulated in software on the Itsy. Although average power usage is much lower on the Itsy, the difference in execution time makes the total energy needed for local recognition substantially higher.

Hybrid recognition completes faster but uses more energy than remote recognition when the client is an Itsy. The energy impact of performing the first stage of recognition locally is greater than the total amount of energy spent waiting for recognition to complete during fully remote execution. This illustrates that tradeoffs between performance and energy conservation exist for common applications—one must carefully consider both the client and its available resources when deciding where to locate functionality.

These results give some evidence that energy-aware adaptation becomes more effective as the ratio between maximum and minimum power usage increases. On the ThinkPad, where this ratio is approximately 1.9, the maximum energy savings due to fidelity reduction and remote execution is only 64–78%. On the Itsy, where this ratio is approximately 6.6, the maximum energy savings

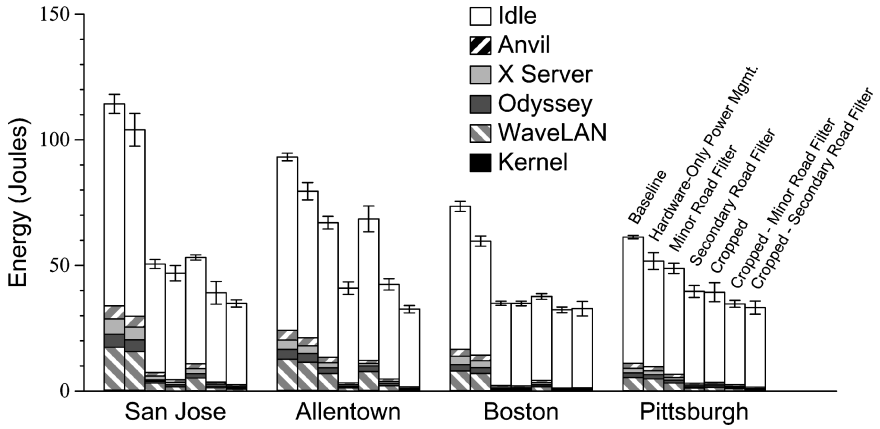


Fig. 12. Energy impact of fidelity for map viewing. This shows the energy used to view U.S. Geological Survey maps of four cities. Each measurement is the mean of 10 trials—the error bars are 90% confidence intervals.

is 92–94%. However, the results are unfortunately biased by the Itsy's lack of floating-point support.

3.5 Map Viewer

3.5.1 *Description.* The third application that we measured was an adaptive map viewer named Anvil. Anvil fetches maps from a remote server and displays them on the client. Fidelity can be lowered in two ways: filtering and cropping. Filtering eliminates fine detail and less important features (such as secondary roads) from a map. Cropping preserves detail, but restricts data to a geographic subset of the original map. The client annotates the map request with the desired amount of filtering and cropping. The server performs any requested operations before transmitting the map.

3.5.2 *Results.* We measured the energy used by the client to fetch and display maps of four different cities. Viewing a map differs from the two previous applications in that a user typically needs a nontrivial amount of time to absorb the contents of a map after it has been displayed. This *think time* should logically be viewed as part of the application's execution since energy is consumed to keep the map visible. In contrast, the user needs negligible time after the display of the last video frame or the recognition of an utterance to complete use of the video or speech application.

Think time is likely to depend on both the user and the map being displayed. Our approach to handling this variability was to use an initial value of 5 s and then perform sensitivity analysis for think times of 0, 10, and 20 s. For brevity, Figure 12 only presents detailed results for the 5-s case; for other think times, we present only the summary information in Figure 13.

The baseline bars in Figure 12 show that most of the energy is consumed while the kernel executes the idle procedure; most of this is background power usage during the 5 s of think time. The shadings on the bars indicate that

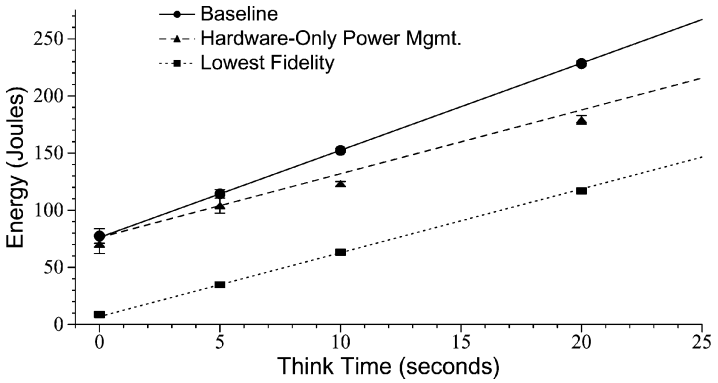


Fig. 13. Effect of user think time for map viewing. This shows how the energy used to view the San Jose map varies with think time. Each measurement is the mean of 10 trials—the error bars are 90% confidence intervals.

network communication is a second significant drain on energy. The comparatively larger confidence intervals for this application result from variation in the time required to fetch a map over the wireless network.

Hardware power management reduces energy consumption by about 9–19% relative to the baseline. Although there is little opportunity for network power management while the map is being fetched, the network can remain in standby mode during think time. Since the disk is never used, it can always remain in standby mode.

The third and fourth bars of each data set show the effect of fidelity reduction through two levels of filtering. One filter omits minor roads, while the more aggressive filter omits both minor and secondary roads. The savings from the minor road filter are 6–51% relative to hardware-only power management. The corresponding figure for the secondary road filter is 23–55%.

The fifth bar of each data set shows the effect of lowering fidelity by cropping a map to half its original height and width. Energy usage at this fidelity is 14–49% less than with hardware-only power management. In other words, cropping is less effective than filtering for these particular maps. Combining cropping with filtering results in an energy savings of 36–66% relative to hardware-only power management. Relative to the baseline, this is a reduction of 46–70%. There is little savings left to be extracted through fidelity reduction since almost all energy is now consumed in the idle state.

After examining energy usage with 5 s of think time, we repeated the above experiments with think times of 0, 10, and 20 s. At any given fidelity, energy usage, E_t increases with think time, t. We expected a linear relationship: $E_t = E_0 + t \cdot P_B$, where E_0 is the energy usage for this fidelity at zero think time and P_B is the background power consumption on the client (5.6 W with the screen on and disk and network in standby mode).

Figure 13 confirms that a linear model is indeed a good fit. This figure plots the energy usage for four different values of think time for three cases: baseline, hardware power management alone, and lowest fidelity combined with hardware power management. The divergent lines for the first two cases show

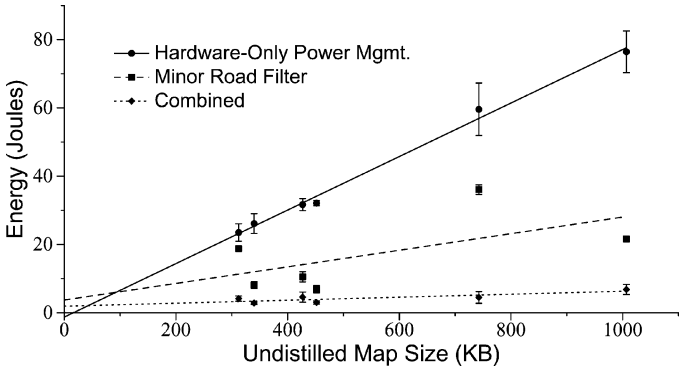


Fig. 14. Predicting map viewer energy use. Each line is the best linear fit for the energy used to view six maps at the same fidelity—the error bars are 90% confidence intervals.

that the energy reduction from hardware power management scales linearly with think time. The parallel lines for the second and third cases show that fidelity reduction achieves a constant benefit, independent of think time. The complementary nature of these two approaches is thus well illustrated by these measurements.

Figure 14 shows energy use as a function of the undistilled map size for three fidelity levels: baseline, minor road filtering, and the combination of cropping and minor and secondary road filtering—the remaining fidelity levels are omitted for clarity. For each fidelity, six data points show the total energy used to fetch and display different city maps, including the four from Figure 12. Since we have already shown that the energy cost of user think time is quite predictable, these results assume zero think time and thereby isolate the variance in the energy cost of fetching and displaying maps.

Visual inspection of the linear fits reveals that energy usage corresponds closely to image size only in the baseline case. However, when simple models do not yield good predictions, often additional information will give more accurate results. Here, the percentage of features omitted by a given filter varies widely from map to map. We modified the map server to store summary information with each map listing the occurrence of each feature type—this allows us to anticipate the effectiveness of a filter. This improves prediction: when the minor-road filter is used, the R^2 value is 99%. However, although the linear fit for the combined fidelity improves, it is still quite poor with an R^2 value of 79%. Although the model can anticipate the effect of filtering, it is still unable to cope with variation in the amount of features removed by cropping.

3.6 Web Browser

3.6.1 *Description.* The fourth application is an adaptive Web browser based on Netscape Navigator. In this application, Odyssey and a distillation server located on either side of a variable-quality network mediate access to Web servers. Requests from an unmodified Netscape browser are routed to a proxy on the client that interacts with Odyssey. After annotating the request with the desired level of fidelity, Odyssey forwards it to the distillation server

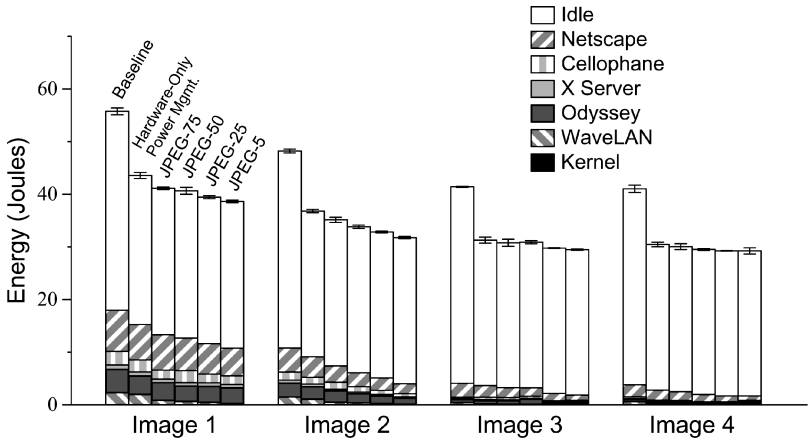


Fig. 15. Energy impact of fidelity for Web browsing. This shows the energy used to display four GIF images from 110 B to 175 KB in size, ordered from right to left above. Each measurement is the mean of ten trials—the error bars show 90% confidence intervals.

which transcodes images to lower fidelity using lossy JPEG compression. This is similar to the strategy described by Fox et al. [1996], except that control of fidelity is at the client.

3.6.2 *Results.* As with the map application, a user needs some time after an image is displayed to absorb its contents. We measured Web browser energy usage with a baseline value for think time of 5 s/image. As expected, sensitivity analysis on think time shows equivalent behavior to that depicted in Figure 13 for the map viewer.

Figure 15 presents measurements of the energy used to fetch and display four GIF images of varying sizes. Hardware power management achieves reductions of 22–26%. The shadings on the first and second bars of each data set indicate that most of this savings occurs in the idle state, probably during think time.

The energy benefits of fidelity reduction are disappointing. The energy used at the lowest fidelity is merely 4–14% lower than with hardware power management alone; relative to baseline, this is a reduction of 29–34%. The maximum benefit of fidelity reduction is severely limited because the amount of energy used during think time (28 J) is much greater than the energy used to fetch and display an image, (2–16 J). Thus, even if fidelity reduction could completely eliminate the energy used to fetch and display an image, energy use would drop only 9–36%.

Although Web fidelity reduction shows little benefit in this study, it may be quite useful in other environments. For example, a more energy-efficient mobile device would use less energy during think time, increasing the possible benefit of fidelity reduction. Similarly, if a high-speed network were unavailable, the energy benefit of distillation would increase.

Figure 16 shows Web browser energy use as a function of the undistilled image size for three fidelities. For each fidelity, seven data points show the total energy used to fetch and display different images, including the four from

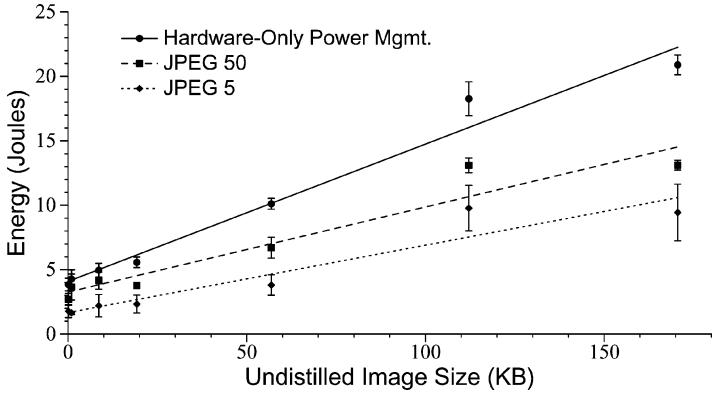


Fig. 16. Predicting Web browser energy use. Each line is the best linear fit for the energy used to fetch and display six Web images—the error bars are 90% confidence intervals.

Figure 15. As with the map application, these results include no user think-time, thereby isolating the variation in energy used to fetch and display images. For this application, simple linear models yield reasonable predictions: the R^2 values for baseline, JPEG-50, and JPEG-5 fidelity are 91%, 93%, and 98%, respectively.

3.7 Effect of Concurrency

How does concurrent execution affect energy usage? One can imagine situations in which total energy usage goes *down* when two applications execute concurrently rather than sequentially. For example, once the screen has been turned on for one application, no additional energy is required to keep it on for the second. One can also envision situations in which concurrent applications interfere with each other in ways that increase energy usage. For example, if physical memory size is inadequate to accommodate the working sets of two applications, their concurrent execution will trigger higher paging activity, possibly leading to increased energy usage. Clearly, the impact of concurrency can vary depending on the applications, their interleaving, and the machine on which they run.

What is the effect of lowering fidelity? The measurements reported in Sections 3.3 to 3.6 indicate that lowering fidelity tends to increase the fraction of energy consumption attributable to the idle state. Concurrency allows background energy consumption to be amortized across applications. It is therefore possible in some cases for concurrency to enhance the benefit of lowering fidelity.

To confirm this intuition, we compared the energy usage of a composite application when executing in isolation and when executing concurrently with the video application described in Section 3.3. The composite application consists of six iterations of a loop that involves the speech, Web, and map applications described in Sections 3.4 to 3.6. The loop consists of local recognition of two speech utterances, access of a Web page, access of a map, and 5 s of think time. The composite application models a user searching for Web and map information using

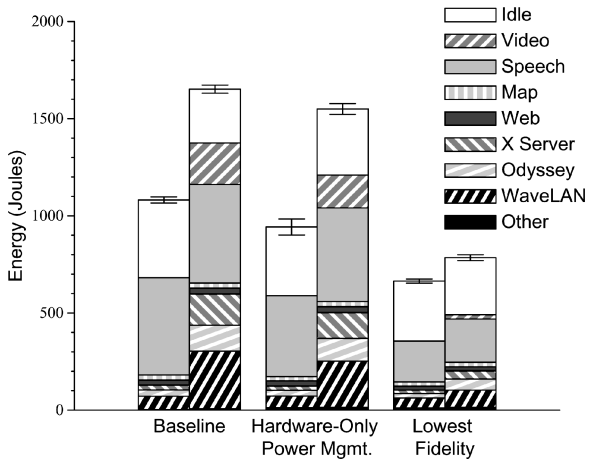


Fig. 17. Effect of concurrent applications. Each data set compares energy usage for the composite application described in Section 3.7 in isolation (left bar) with total energy usage when a video application runs concurrently (right bar). Each measurement is the mean of five trials—the error bars are 90% confidence intervals.

speech commands, while the video application models a background newsfeed. This experiment takes between 80 and 160 s.

Figure 17 presents results for three cases: baseline, hardware-only power management, and minimal fidelity. In the first two cases, all applications ran at full fidelity; in the third case, all ran at lowest fidelity. For each data set, the left bar shows energy usage for the composite application in isolation, while the right bar shows energy usage during concurrent execution.

For the baseline case, the addition of the video application consumes 53% more energy. But with hardware power management, it consumes 64% more energy. This difference is due to the fact that concurrency reduces opportunities for powering down the network and disk. For the minimum fidelity case, the second application only adds 18% more energy. The significant background power usage of the client, which limits the effectiveness of lowering fidelity, is amortized by the second application. In other words, for this workload, concurrency does indeed enhance the energy impact of lowering fidelity.

3.8 Discussion

Our primary goal in performing this study was to determine whether lowering fidelity yields significant energy savings. The results of Sections 3.3 to 3.6 confirm that such savings are indeed available over a broad range of applications relevant to mobile computing. Further, those results show that lowering fidelity can be effectively combined with hardware power management. Section 3.7 extends these results by showing that concurrency can magnify the benefits of lowering fidelity. Finally, energy savings through fidelity reduction are predictable—an adaptive system that constructs simple models of energy use will often be able to accurately anticipate the effect of fidelity changes.

Application	Think Time (s)	Hardware Power Mgmt.	Fidelity Reduction	Combined
Video	N/A	0.90–0.91	0.84–0.84	0.65–0.65
Speech	N/A	0.66–0.67	0.22–0.36	0.20–0.31
Map	0	0.80–1.01	0.06–0.13	0.07–0.18
	5	0.81–0.91	0.38–0.67	0.31–0.54
	10	0.74–0.84	0.53–0.77	0.42–0.58
	20	0.76–0.78	0.69–0.89	0.51–0.67
Web	0	0.85–1.06	0.40–0.75	0.32–0.54
	5	0.74–0.78	0.88–0.97	0.66–0.71
	10	0.75–0.78	0.93–0.98	0.70–0.74
	20	0.74–0.77	0.96–0.99	0.72–0.73

Fig. 18. Energy impact of fidelity. Each entry shows the minimum and maximum measured energy consumption on the IBM ThinkPad 560X for four data objects. The entries are normalized to baseline measurements of full fidelity objects with no power management.

While this study examined only cases where application source code is available, further work [Flinn et al. 2001] has shown that proxy-based solutions can add energy-awareness to closed-source applications. Using de Lara's Puppeteer as an adaptive platform, we showed that fidelity reduction can reduce the energy needed to load and edit PowerPoint documents stored on a remote server by up to 49%.

At the next level of detail, Figure 18 summarizes the results of Sections 3.3 to 3.6. Its key messages are:

—*There is significant variation in the effectiveness of fidelity reduction across data objects.* The reduction can span a range as broad as 29% (0.38–0.67 for the map viewer at think time 5). The video player is the only application that shows little variation across data objects. As mentioned above, this variation is often quite predictable with simple linear models.

—*There is considerable variation in the effectiveness of fidelity reduction across applications.* Holding think time constant at 5 s and averaging across data objects, the energy usage for the four applications at lowest fidelity is 0.84, 0.28, 0.51, and 0.93 relative to their baseline values. The mean is 0.64, corresponding to an average savings of 36%.

—*Combining hardware power management with lowered fidelity can sometimes reduce energy usage below the sum of the individual reductions.* This is seen most easily in the case of the video application, where the last column is 0.65 rather than the expected value of 0.74. Intuitively, reducing fidelity decreases hardware utilization, thereby increasing the opportunity for hardware power management.

4. GOAL-DIRECTED ADAPTATION

Our study shows that energy-aware applications can substantially reduce the energy usage of mobile computers. Yet, it provides no guidance as to how applications should balance the competing goals of energy conservation and fidelity. In this section, we explore whether the operating system can advise applications how to strike the balance.

A mobile user often has a reasonable estimate of how long a battery needs to last—for example, the expected duration of a commute, meeting, or flight. Given such an estimate, one can ask whether energy-aware adaptation can be exploited to yield the desired battery life. We explored this question by modifying the Odyssey platform for mobile computing [Noble et al. 1997] to monitor energy supply and demand and to use this information to direct application adaptation to meet a user-specified goal for battery duration. We call this process *goal-directed adaptation*.

Today, operating systems report expected battery lifetime to the user and provide a number of parameters that can be adjusted to extend battery lifetime, that is, display brightness, processor speed, and device timeouts. The user must continually monitor battery state and adjust these parameters in order to ensure that their battery lasts for the needed duration. Goal-directed adaptation fundamentally inverts this process by making energy management the responsibility of the operating system. The user specifies only desired battery lifetime—the system makes the correct tradeoffs between quality and energy conservation that ensure that the battery lasts for the specified duration.

While implementing goal-directed adaptation, our most important design goal was ensuring that Odyssey met the specified time goal whenever feasible. When a user specifies an infeasible duration, one so large that the available energy is inadequate even if all applications run at lowest fidelity, Odyssey should detect the problem and alert the user as soon as possible.

An important secondary goal was providing the best user experience possible. This translates into two requirements: first, applications should offer as high a fidelity as possible; second, the user should not be jarred by frequent adaptations. Goal-directed adaptation balances these opposing concerns by striving to provide high average fidelity while using hysteresis to reduce fidelity changes.

Next, we describe the user and application interfaces that support goal-directed adaptation. We then describe how we have modified Odyssey to periodically determine the residual energy available in the battery, predict future energy demand, and decide what fidelity each application should use.

4.1 User and Application Interfaces

Figure 19 shows the user interface for goal-directed adaptation; we have designed it to be as simple as possible so as to avoid unnecessary user distraction. The user specifies the goal for battery duration by adjusting the slider in the middle of the dialog. When conditions change, the user readjusts the slider to specify a new battery goal. As time passes and the battery drains, the slider moves to the left to express the change in expected remaining battery lifetime. While current operating systems provide similar dialogs, they are output-only; they do not provide users with the ability to change the expected battery lifetime.

In this paper, we describe only the relevant portion of Odyssey's API—Narayanan et al. [2000] provided a detailed description of the compete API.

Fig. 19. Interface for goal-directed adaptation.

The fundamental unit of dialog between Odyssey and applications is an *operation*: a discrete unit of work performed at one of possibly many fidelities. For example, each application described in Section 3 performs one basic operation: the video player displays videos, the speech recognizer recognizes utterances, the map viewer displays maps, and the Web browser displays images.

When an application first starts, it calls `register_fidelity` to describe the types of operations that it may execute and the fidelities at which they may be performed. Before executing each operation, the application calls `begin_fidelity_op`. Odyssey returns the fidelity at which the operation should be performed. When the operation completes, the application calls `end_fidelity_op`. This model is *cooperative*: applications are free to specify what fidelities they are willing to support, and Odyssey does not enforce its choice of fidelities.

4.2 Determining Residual Energy

Odyssey determines residual energy by measuring the amount of charge in the battery. Many modern mobile computers ship with a Smart Battery [SBS Implementers Forum 1998], a gas gauge chip that reports detailed information about battery state and power usage. On such platforms, Odyssey periodically queries the Smart Battery through ACPI [Intel, Microsoft, and Toshiba 1998] or a machine-dependent interface. For example, the Itsy v2.2 [Hamburgen et al. 2001] contains a Dallas Semiconductor DS2437 Smart Battery chip [Dallas Semiconductor Corp. 1999]—on this platform, Odyssey queries battery status once per second by performing an `ioctl` on a device driver.

While the above methods are best for deployed systems, they may not be ideal in laboratory settings. Older mobile computers such as the IBM 560X lack the necessary hardware support for measuring battery capacity. To facilitate evaluation, Odyssey can use a *modulated* energy supply. At the beginning of evaluation, the initial value for the modulated supply is specified. As the evaluation proceeds, a digital multimeter samples actual power usage and transmits the samples to Odyssey. Odyssey decrements the modulated energy supply and reports when it has expired. This approach assumes an ideal battery. In practice, batteries are nonlinear—as power draw increases, the total energy that can be

extracted from a battery decreases. In all of our experiments, fidelity reduction decreases average power usage—in such cases, this simplifying assumption slightly understates the benefits of fidelity reduction.

4.3 Predicting Future Demand

To predict energy demand, Odyssey assumes that future behavior will be similar to recently-observed behavior. It uses smoothed observations of present and past power usage to predict future power use. This approach is in contrast to requiring applications to explicitly declare their future energy usage—an approach that places unnecessary burden on applications and is unlikely to be accurate.

Odyssey periodically samples power levels using the interfaces described in the previous section. It smooths these samples using an exponential weighted average of the form:

$$\text{new} = (1 - \quad)(\text{this sample}) + (\quad)(\text{old}), \tag{2}$$

where is the gain, a parameter that determines the relative weights of current and past power usage. Odyssey multiplies this estimate of future power use by the time remaining until the goal to predict future energy demand.

Odyssey varies as energy drains, thus changing the tradeoff between agility and stability. When the goal is distant, Odyssey uses a large . This biases adaptation toward stability by reducing the number of fidelity changes—there is ample time to make adjustments later, if necessary. As the goal nears, Odyssey decreases so that adaptation is biased toward agility. Applications must respond more rapidly since the margin for error is small.

Currently, Odyssey sets so that the half-life of the decay function is approximately 10% of the time remaining until the goal. For example, if 30 min remain, the present estimate will be weighted approximately equal to more recent samples after 3 min. The value of 10% is based upon a sensitivity analysis discussed in Section 4.7.2.

4.4 Triggering Adaptation

When predicted demand exceeds residual energy, Odyssey advises applications to reduce their energy usage. Conversely, when residual energy significantly exceeds predicted demand, applications are advised to increase fidelity.

The amount by which supply must exceed demand to trigger fidelity improvement is indicative of the level of hysteresis in Odyssey's adaptation strategy. This value is the sum of two empirically derived components: a variable component, 5% of residual energy, and a constant component, 1% of the initial energy supply. The variable component reflects the bias toward stability when energy is plentiful and toward agility when it is scarce; the constant component biases against fidelity improvements when residual energy is low. As a guard against excessive adaptation due to energy transients, Odyssey caps fidelity improvements at a maximum rate of once every 15 s.

To meet the specified goal for battery duration, Odyssey tries to keep power demand within the zone of hysteresis. Whenever power demand leaves this

zone, adaptation is necessary. Odyssey then determines which applications should change fidelity, as well as the magnitude of change needed for each application. Ideally, the set of fidelity changes should modify power demand so that it once again enters the zone of hysteresis.

Often, there will be many possible adaptations that yield the needed change in power demand. In these cases, Odyssey refers to an *adaptation policy* to choose an alternative. Although many policies are possible, Odyssey currently supports two: an incremental policy described in the next section, and a history-based policy described in the following section.

4.5 Incremental Adaptation Policy

When calling `register_fidelity`, each application specifies a list of supported fidelities. Using the incremental policy, Odyssey maintains a per-application variable that specifies which fidelity should be used by the application. It returns this value whenever the application calls `begin_fidelity_op`.

Odyssey lets the user specify the relative priority of applications. When predicted demand exceeds measured supply, Odyssey degrades the fidelity of the lowest-priority application by changing its fidelity variable to the next lower supported fidelity. Once the lowest-priority application reaches its lowest fidelity level, Odyssey degrades the next lowest-priority application. Fidelity upgrades occur in reverse.

After an application adapts, Odyssey verifies that the change in power demand is sufficient. It resets the estimate of power demand so that it is in the middle of the zone of hysteresis—this represents an optimistic assumption that the change in fidelity will prove sufficient. As Odyssey takes new power measurements, the estimate of power demand will either stay within the zone of hysteresis, indicating that the change in fidelity was correct, or stray outside the bounds, indicating that additional adaptation is necessary.

Our experimental results show that the incremental policy usually does an excellent job of meeting user-specified goals for battery lifetimes. However, it does possess limitations that make the design of energy-aware applications more challenging. First, an application must ensure that each lower fidelity uses less energy per unit of work than all higher fidelities. Second, some care must be exercised in choosing the number of supported fidelities. If too many are specified, Odyssey may be very sluggish in adapting to changes in power demand because it must incrementally traverse a large number of fidelities. If too few are specified, the system may oscillate frequently between two adjacent fidelities that have a large difference in power demand. These observations led us to also develop a history-based policy.

4.6 History-Based Policy

Using the history-based adaptation policy, Odyssey measures and logs the energy that applications use at different fidelities. It learns functions which relate application fidelity and energy usage. When Odyssey needs to change system power usage to meet battery goals, it consults these functions and selects the adaptation that it predicts will yield the needed change in power usage.

4.6.1 *Recording Application Energy History.* Odyssey measures the energy supply of the mobile computer when the application calls `begin_fidelity_op` and `end_fidelity_op` for each operation—these on-line measurements are performed using one of the methods described in Section 4.2. The difference between these two values gives the total system energy expended during the execution of the operation. Odyssey records operation energy usage in an application-specific log, along with the operation start time, end time, and fidelity. Odyssey also records the value of any parameters that affect the complexity of the operation—for example, the length of a speech utterance or the size of a Web image. The value of these operation-specific parameters are specified when an application calls `begin_fidelity_op`. Odyssey buffers the log in memory and batch writes data to persistent storage, making this information available across multiple invocations of an application.

If more than one operation executes simultaneously, Odyssey does not try to differentiate their individual energy impact. Potentially, it could apportion energy usage according to the percentage of CPU, disk, network, and other resources used by each operation. However, this method requires detailed resource accounting, as well as per-platform calibration of the energy costs of resource usage. We have therefore chosen the simpler approach of excluding from the history log any operation which overlaps with another simultaneous operation. This sacrifices some data, but ensures that only accurate data is logged.

4.6.2 *Learning from Application Energy History.* Odyssey models application energy usage as a function of fidelity and input parameters. As Section 3 shows, simple linear models provide a good fit for many applications—we support this common case with default library routines. We also provide a modular interface that enables more complex applications to supply application-specific prediction functions. In practice, we have found the default routines to be sufficient for all applications that we have studied to date.

The default prediction library treats each dimension of fidelity and each parameter as a separate dimension in a multidimensional space. It uses multiple variable linear regression to model continuous dimensions, and binning to model discrete dimensions. For example, the video recognizer has a continuous input parameter, the length of the video clip, and two discrete dimensions, compression level and window size. Odyssey uses linear regression to generate a linear model for each combination of compression and window size—these six models correspond to the lines in Figure 8.

4.6.3 *Benefits of Application Energy History.* With the history-based policy, applications no longer need to carefully choose fidelities so that each specified fidelity uses less energy than all higher fidelities. Figure 20 demonstrates why this is important. Each line shows the energy usage of the Web browser described in Section 3.6 at a specific fidelity. Energy use varies significantly with the uncompressed size of the image being displayed. The highest fidelity corresponds to the baseline case where no distillation is performed; the other fidelities correspond to image distillation to JPEG quality levels of 99 and 95. Data points show measured energy values for two image sizes: 720 KB and

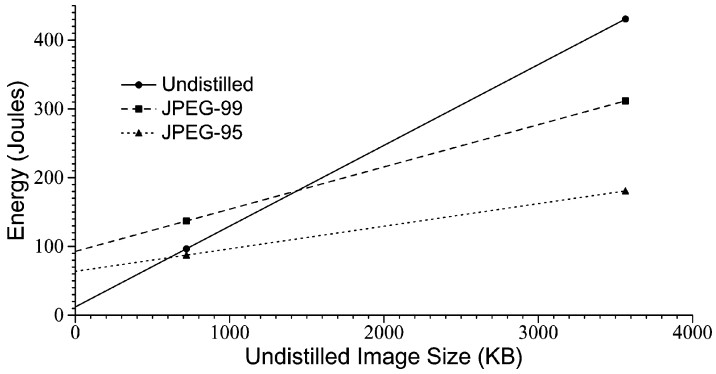


Fig. 20. Variation in Web energy use. The three lines show how Web browser energy usage changes with image size for three fidelities; the data points are measured energy usage for two specific image sizes.

3.5 MB. For large images, the highest fidelity uses more energy since it transmits the most data over the wireless network. However, for small images, the lower fidelities use more energy because image distillation introduces a significant latency which results in the client waiting longer for the image to arrive. Thus, it is impossible to specify a static ordering of these three fidelities such that energy decreases with fidelity across all image sizes. The incremental policy would incorrectly switch from baseline to JPEG-99 when small images are being displayed and a reduction in power usage is needed—this transition would actually *increase* power demand.

The history-based policy introduces a layer of indirection in the form of a global feedback parameter, c, that represents how critical energy conservation is at the current moment. The value of this parameter ranges from 0 to 1 and represents the relative power reduction that each application should provide through fidelity reduction. For example, when c is 0.1, each application is requested to decrease its energy usage to no more than 90% of its baseline (maximum) energy consumption. When an application calls `begin_fidelity_op`, Odyssey consults its energy model and selects the best fidelity that yields at least a 10% reduction in energy usage. In the event that no supported fidelity provides the desired energy reduction, Odyssey selects the fidelity that maximizes energy conservation.

4.6.4 *Using Energy History to Improve Agility.* The history-based policy is more agile than the incremental policy because it can directly jump to an appropriate fidelity level when adaptation is needed. The incremental policy must traverse each intermediate fidelity and pause after each adaptation to assess how much power demand has changed. When a large number of fidelities are specified, the incremental policy can potentially be too sluggish in reacting to changes in power demand.

How does the history-based policy select an appropriate fidelity level? First, it calculates the needed change in power usage. During normal operation, predicted energy demand remains within a zone of hysteresis with a value no

greater than the energy supply and no lower than the 95% of the current supply minus 1% of the initial energy supply. When the current power demand, $P_{current}$, strays outside this zone, adaptation is needed.

The ideal power demand is precisely in the middle of the zone of hysteresis. If S is the current energy supply, t the time remaining to the goal, and S_i the initial energy supply, the ideal power demand, P_{ideal}, is:

$$P_{ideal} = (S + (0.95\ S - 0.01\ S_i))/2t. \quad (3)$$

Odyssey calculates the needed change in power as $P_{ideal} - P_{current}$. Next, it predicts the ideal value of c, c_{ideal}, that will produce a future power drain of P_{ideal}. Finally, it sets the value of c directly to c_{ideal}. When applications subsequently call `begin_fidelity_op`, Odyssey will choose fidelity levels that yield the desired amount of energy conservation.

Unfortunately, Odyssey cannot calculate c_{ideal} directly. The feedback parameter, c, represents only the *requested* change in each application's individual power usage. Some applications may not have sufficiently low fidelities to provide the requested power reduction. Additionally, energy-aware applications account for only a portion of total system power usage; background power consumption such as the screen backlight and the activities of applications that are not energy-aware account for the rest. Thus, an $x\%$ reduction in c does not guarantee that there will be an $x\%$ reduction in system power usage; often the resulting change is much less.

Odyssey uses its history of application energy usage to estimate the relationship between c and system power usage. It maintains a list of recently executed operations that includes associated input parameters and fidelity decisions. For a given value of c, it calculates the fidelity that it would have chosen for each logged operation. It then uses application-specific energy models to predict how much *dynamic energy* each operation would have used at the selected fidelity. Dynamic energy is the energy needed to perform an operation above and beyond what would have been needed to execute the kernel idle procedure for the same time period.

Odyssey then constructs a hypothetical waveform that captures its prediction of how the mobile computer's power usage would have changed over time for the specified value of c. The waveform reflects the background power usage of the machine and the predicted dynamic energy usage of each operation performed in the past. Finally, Odyssey calculates power demand by applying the exponential smoothing function shown in Equation (2) to the waveform.

Figure 21(a) shows a hypothetical example that helps illustrate this process. Here, Odyssey is predicting power demand for $c = 0.1$. There are four recently executed operations. Odyssey predicts what their energy usage would have been if c had been 0.1. For example, it predicts that operation 1 would have used 3 W. Odyssey then generates the waveform that represents predicted power usage over time. Total power usage at any given time is the sum of background power usage and the predicted dynamic power usage of each currently executing operation. In this figure, the total power usage waveform is given by the area under all shaded regions. Using Equation (2), Odyssey would calculate power demand to be approximately 3.81 W in this example.

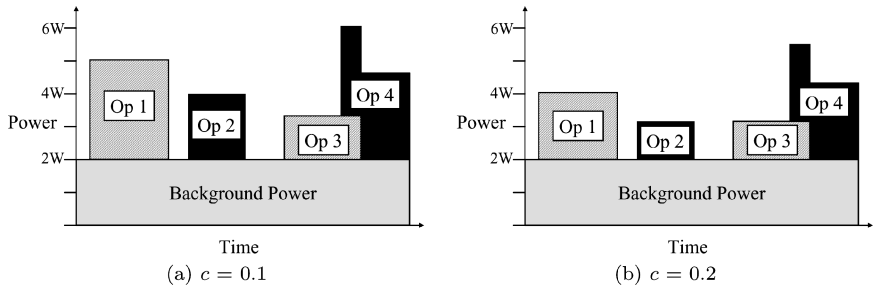


Fig. 21. Example of history replay.

Figure 21(b) shows the same calculation for $c = 0.2$. Odyssey selects lower fidelity levels for each operation. Consequently, the dynamic energy usage of each operation is lower than in Figure 21(a). For $c = 0.2$, Odyssey would predict power demand to be about 3.18 W.

Odyssey performs a binary search to determine c_{ideal}. That is, it calculates the new value of c that would place power demand squarely in the middle of the zone of hysteresis. It then changes the current value of the feedback parameter to c_{ideal} and resets its estimate of the current power demand. Note that this prediction need not always be correct; the feedback in the system will eventually adjust for any errors.

4.7 Validation

4.7.1 *Benefit of Goal-Directed Adaptation.* To explore the benefit of goal-directed adaptation, we executed the two energy-aware applications described in Section 3.7: a composite application involving speech recognition, map viewing, and Web access, run concurrently with a background video application. To obtain a continuous workload, the composite application executes every 25 s. Thus, the amount of work (number of recognitions, image views, and map views) is constant over time.

The video application supports four fidelities: full-quality, Premiere-B, Premiere-C, and the combination of Premiere-C and reduction of the display window. The speech recognition component of the composite application supports full and reduced quality recognition on the local machine. The map component has four fidelities: full-quality, minor road filtering, secondary road filtering, and the combination of secondary road filtering and cropping. The Web component supports the five fidelities shown in Figure 15. Odyssey uses the incremental adaptation policy; we assign speech the lowest priority, with video, map, and Web having successively higher priorities.

Client applications execute on the IBM 560X and communicate with servers using a 900-MHz, 2-Mb/s wireless network. To isolate the performance of goal-directed adaptation, Odyssey uses a modulated energy supply.

At the beginning of each experiment, we provided Odyssey with an initial energy value and a time goal. We then executed the applications, allowing them to adapt under Odyssey's direction until the time goal was reached or the residual energy dropped to zero. The former outcome represents a successful

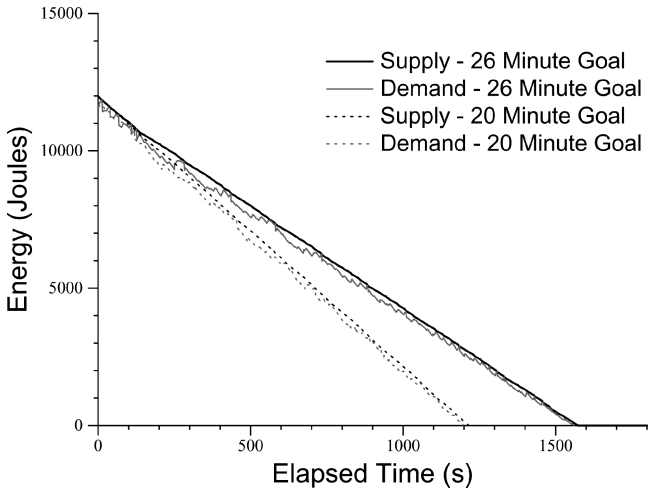


Fig. 22. Change in energy supply and demand. Odyssey meets user-specified goals for battery durations of 20 and 26 min while running the composite and video applications described in Section 3.7.

completion of the experiment, while the latter represents a failure. We noted the total number of adaptations of each application. We also noted the residual energy at the end of the experiment—a large value suggests that Odyssey may have been too conservative in its adaptation and that average fidelity could have been higher.

All experiments used a 12,000-J modulated energy supply. This lasts 19:27 min when applications operate at highest fidelity, and 27:06 min at lowest fidelity. The difference between the two values represents a 39.3% extension in battery life. We deliberately chose a small initial energy value so that we could perform a large number of experiments in a reasonable amount of time. The value of 12,000 J is only about 14% of the nominal energy in the IBM 560X battery. Extrapolating to full nominal energy, the workload would run for 2:18 hours at highest fidelity, and 3:13 hours at lowest fidelity.

Figures 22 and 23 show detailed results from two typical experiments: one with a 20-min goal, and the other with a 26-min goal. Figure 22 shows how the supply of energy and Odyssey's estimate of future demand change over time. In both trials, Odyssey meets the specified goal for battery duration. In addition, once the time goal is reached, residual energy is quite low. The graph also confirms that estimated demand tracks supply closely. The most visible difference between the two trials is the slope of the supply and demand lines. After an initial adaptation period of approximately three minutes, Odyssey adjusts power usage in the 26-min trial in order to extend battery lifetime.

The four graphs of Figure 23 show how the fidelity of each application varies. For the 20-min goal, the high-priority Web and map applications remain at full fidelity throughout the experiment; the video degrades slightly; and speech runs mostly at low fidelity. For the 26-min goal, the highest-priority Web application

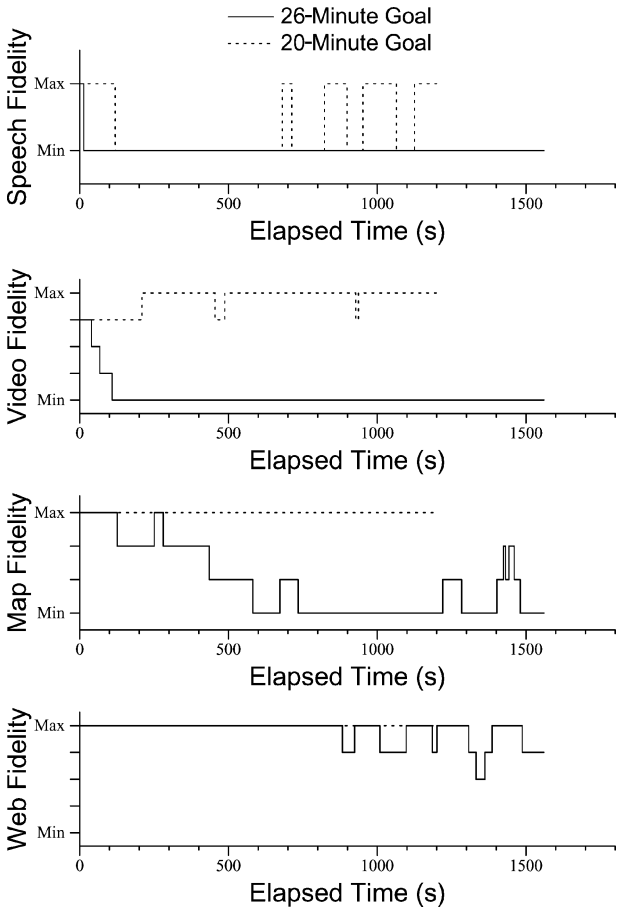


Fig. 23. Change in application fidelity. The four graphs show how each application adapts when 20- and 26-min goals for battery durations are specified.

runs mostly at the highest-fidelity, while the other three applications run mostly at their lowest fidelities. For both goals, application behavior is more stable at the beginning and exhibits greater agility as energy drains.

Figure 24 summarizes the results of five trials for each time goal of 20, 22, 24, and 26 min. These results confirm that Odyssey is doing a good job of energy adaptation. The desired goal was met in every trial. In all cases, residual energy was very low: the largest average residue is still only 1.2% of the initial energy value. The average number of adaptations by applications is generally low, but there are some cases where it is high. This is an artifact of the small initial energy value, since the system is designed to exhibit greater agility when energy is scarce.

All values used in this experiment are feasible battery lifetimes. If a value greater than 27:06 is selected, Odyssey would clearly fail to meet the goal for battery lifetime. In this situation, Odyssey should notify the user as soon as possible using the interface in Figure 19. Using the incremental policy, Odyssey

Goal (s)	Goal Met	Residue		Number of Adaptations			
		Energy (J)	Time (s)	Speech	Video	Map	Web
1200	100%	145.2 (25.3)	15.3 (1.9)	10.8 (1.6)	11.0 (4.0)	0.4 (0.9)	0.0 (0.0)
1320	100%	107.5 (61.5)	12.9 (7.2)	2.8 (0.4)	28.2 (5.2)	1.6 (2.6)	0.0 (0.0)
1440	100%	101.2 (22.3)	13.0 (4.5)	5.0 (7.9)	22.6 (9.8)	9.6 (3.8)	1.2 (1.8)
1560	100%	60.2 (28.7)	8.7 (5.9)	1.0 (0.0)	6.0 (2.8)	15.4 (4.6)	7.6 (5.9)

Fig. 24. Summary of goal-directed adaptation. The second column shows the percentage of trials in which the energy supply lasts for at least the specified duration. The next two columns show the average residual energy at the end of the experiment. The remaining columns show the average number of adaptations performed by each application. Each entry represents the mean of five trials with standard deviations given in parentheses.

Half-Life	Goal Met	Residue (J)	Adaptations
0.01	100%	204.6 (17.7)	93.6 (3.7)
0.05	100%	124.1 (38.0)	33.2 (4.0)
0.10	100%	129.2 (21.6)	14.6 (5.4)
0.15	80%	97.6 (22.2)	6.8 (2.9)

Fig. 25. Sensitivity to half-life. The first column is the half-life value used for smoothing. The second column shows the percentage of trials in which the energy supply lasts for at least the specified duration. The next column shows the residual energy at the end of the experiment. The final column shows the number of adaptations performed. Each entry is the mean of five trials with standard deviation given in parentheses.

would trigger successive adaptations until all applications operate at their minimum fidelity—it would then notify the user when the goal still proved infeasible. Using the history-based policy, Odyssey would notify the user as soon as it realized that no set of adaptations would be sufficient to produce the needed change in power usage. This issue is explored further in Section 4.7.4.

4.7.2 *Sensitivity to Half-Life.* We next examined how changing the half-life parameter used to calculate the gain, , affects system performance. In each experiment, we displayed a half-hour video clip and specified that the modulated 13,000-J energy supply should last for the entire video.

Figure 25 summarizes the results of five trials each for several half-life values. A half-life of 1% is clearly too unstable—the system produces the largest residue with this value, and the video player adapts excessively. For larger values, the system is more stable. However, with a 15% half-life, the system is insufficiently agile, failing to meet the goal in one of the five trials. We have also encountered similar results in less detailed analysis of the speech, map, and Web applications. This led us to use a 10% half-life.

4.7.3 *Dynamic Workloads.* The short duration of the previous experiments allowed us to explore the behavior of Odyssey for many parameter combinations. Having established the feasibility of goal-directed adaptation, we then ran a small number of longer duration experiments with more dynamic workloads to confirm its benefits in more realistic scenarios.

We used a simple stochastic model to construct an irregular, dynamic workload. During any given minute, each of four applications (video, speech, map, and Web) may independently be active or idle. An active application executes

Trial	Goal Met	Residual Energy (J)	Number of Adaptations			
			Speech	Video	Map	Web
1	Yes	345	1	5	5	1
2	Yes	381	1	10	7	11
3	Yes	2486	8	13	5	0
4	Yes	554	2	10	6	8
5	Yes	464	5	6	14	0

Fig. 26. Longer-duration results. The second column shows whether the energy supply lasts for at least the specified duration. The next column shows the residual energy at the end of the experiment. The remaining columns show the number of adaptations performed.

a fixed workload for one minute; that is, the video application shows a 1-min video. After each minute, there is a 10% chance of switching states; that is, an active application stays active, and an idle one stays idle, with probability 0.9.

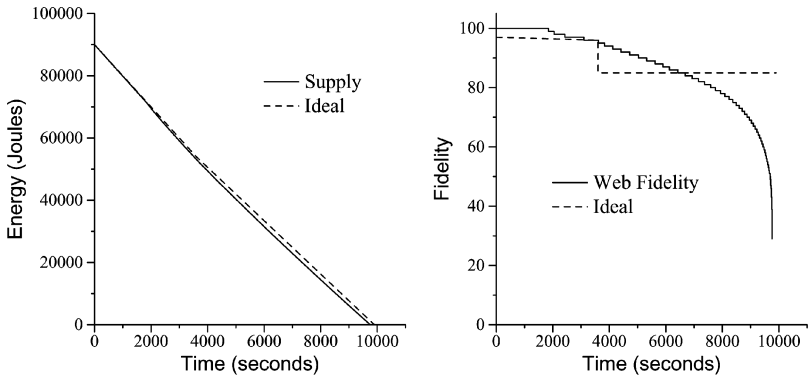
We began each experiment with an energy supply of 90,000 J, roughly matching a fully-charged ThinkPad 560X battery. We specified an initial time duration of 2:45 h, but extended this goal by 30-min at the end of the first hour. This change reflects the possibility that a user may modify the estimate of how long the battery needs to last.

Figure 26 presents the results of five trials of this experiment, each generated with a different random number seed. In every case, Odyssey succeeded in meeting its time goal. Four of the five trials ended with a residual energy that was less than 1% of the initial supply. Only in one trial (trial 3) was the residue noticeably higher (2.8%), implying that Odyssey was too conservative in its adaptation.

In spite of the bursty workload, Figure 26 shows fewer adaptations than Figure 24, which had a steady workload. This is due to two interactions between the hysteresis strategy and the longer-duration goal. First, the zone of hysteresis is larger since it is proportional to the energy supply. Second, smoothing is more aggressive when the goal is distant. Combined, these factors cause Odyssey to ignore minor fluctuations in power usage except toward the end of each trial.

4.7.4 *Benefit of History-Based Policy.* We validated the benefit of the history-based policy with a scenario that models the actions of a user browsing digital photographs from an album stored on a remote Web server. The user loads five images/min—each image is 720 KB in size. We first executed the scenario using the incremental policy; we then executed it using the history-based policy.

The Web application performs one type of operation: fetching an image from the server. JPEG quality is the sole dimension of fidelity; it is specified as a discrete dimension ranging from 5 to 100. This experiment is designed to bring out the flaws in the incremental adaptation policy. There are 96 discrete fidelity levels, each of which produces only a small change in power usage. Thus, the incremental policy must make many adaptations to produce even a small change in application power usage. Fidelity 100 corresponds to loading an undistilled image—the remaining fidelities correspond to distilling an image to the equivalent JPEG quality level before fetching it from the remote server.



(a) Energy drain (b) Fidelity change

Fig. 27. Incremental policy.

Although image size is an input parameter for the operation, all images are roughly the same size.

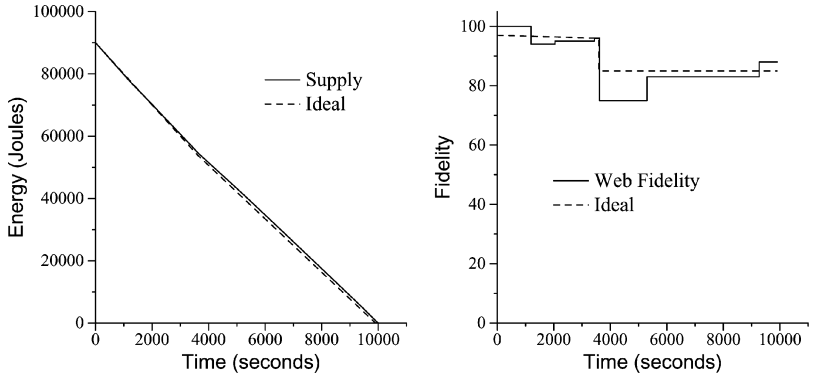
We first obtained a detailed history of energy usage by performing 4000 image loads. This data revealed that fidelities 97–99 use more energy than the baseline fidelity of 100 for the target image size—this is because distillation overhead outweighs savings due to reduction in bytes transmitted over the network. Since fidelities 97–99 are inferior to fidelity 100 in both quality and energy conservation, they should never be chosen by a correct adaptation policy.

We next executed five trials for each adaptation policy. We used a modulated 90,000-J energy supply, roughly equivalent to the amount of energy in the ThinkPad's battery. We specified an initial time goal of 2.5 h. After 1 h of execution, we specified that the battery should last an additional 15 min.

Figure 27 shows results from one typical trial using the incremental policy. The solid line in Figure 27(a) shows how energy supply changes over time. The dashed line shows the rate of drain that would be achieved by an ideal adaptive policy. The ideal policy uses knowledge of future activity to choose the highest constant fidelity that meets the specified goal for battery duration. However, it does not anticipate changes in the specified goal for battery duration—hence, there is a change in fidelity at the 1-h mark.

In the trial, the rate of drain for the incremental policy tracks the ideal fairly closely for the first hour, although energy is used at a slightly higher rate during this time period. Once the user requests an additional 15 min of battery life, the two lines diverge more noticeably. Odyssey uses too much energy after the adjustment and is never able to recover fully. The energy supply expires almost 3 min too early.

Figure 27(b) gives more insight into why Odyssey misses the time goal. After the user changes the goal at 3600 s, Web fidelity decreases exponentially. Initially, changes in fidelity are relatively infrequent because the goal is distant. As the goal nears, Odyssey is more agile and fidelity changes are more frequent. Web fidelity remains higher than ideal for a significant portion of time: from 3600 to 6435 s, causing the laptop to expend too much energy. Even though



(a) Energy drain (b) Fidelity change

Fig. 28. History-based policy.

Adaptation Policy	Goal Met?	Expiration Time (s)		Number of Adaptations	
Incremental	0%	+79.7	(9.7)	56.8	(11.2)
History-Based	100%	-146.6	(11.2)	7.8	(1.3)

Fig. 29. Benefit of energy history.

Odyssey tries to compensate by lowering fidelity for the remainder of the trial, it is unable to conserve enough energy to meet the specified goal for battery duration.

It is also interesting to note that the incremental adaptive policy chooses Web fidelities 97–99 during the time period lasting from 1845 to 3113 s. Without application history, it cannot know that these fidelities are clearly inferior to fidelity 100. This accounts for the slight overconsumption of energy during the first hour of the trial.

Figure 28 shows results from one typical trial using the history-based policy. The rate of drain of the energy supply tracks the ideal quite closely. Odyssey maintains a slight surplus throughout the trial and meets the goal for battery duration. Once the goal is reached, residual energy is only 1.0% of the initial energy supply.

For the first hour, Odyssey chooses fidelities that are close to ideal. It also avoids the clearly inferior fidelities from 97 to 99. When the user respecifies the goal, Odyssey immediately decreases fidelity. The initially chosen value is slightly less than ideal, reflecting Odyssey's conservative bias. However, Odyssey soon detects the discrepancy and readjusts the fidelity. In contrast to the previous scenario, Odyssey is much more agile in converging upon an acceptable fidelity level.

Figure 29 summarizes the results of five trials for each policy. The history-based policy always meets the time goal. When the incremental policy is used, the energy supply expires too early. Further, the average number of adaptations is much smaller with the history-based policy because Odyssey converges much more quickly upon acceptable fidelities. Of course, the incremental policy

would work fine in this scenario if the Web application carefully chose the type and number of fidelities it supported. In contrast, one of the main benefits of the history-based policy is that it is self-tuning—it requires little application support in order to yield acceptable results.

Given these results, our current approach is to use the history-based policy whenever Odyssey has sufficient data to construct an energy model. This means that an application initially uses the incremental policy while Odyssey monitors and persistently logs its energy usage as described in Section 4.6.1. It may take some time before sufficient data is logged to allow construction of an energy model; for example, the history-based experiments in this section assume 4000 previous image loads. However, once the log is sufficient, the application will always be able to use the history-based policy in the future. Currently, the decision of when to switch to the history-based policy is made manually—however, we hope to develop methods that automatically decide when to switch based upon the amount of data that has been logged.

4.7.5 *Overhead.* The power overhead imposed by goal-directed adaptation is the sum of the overhead of measuring power usage and the overhead of using these measurements to predict energy demand. The measurement overhead is quite low. Most Smart Battery solutions can provide power measurements at the frequency Odyssey requires using less than 10 mW [Dallas Semiconductor Corp. 1999; USAR Systems, Inc. 1999]. Odyssey's prediction overhead is only 4 mW. Therefore, the total power overhead is less than 14 mW, or less than 0.25% of the background power consumption of the IBM 560X.

The use of application history incurs additional overhead when Odyssey determines a new value for the feedback parameter c. During the five trials, Odyssey took an average of 0.83 s and 6.7 J to calculate a new value for c. However, since this calculation is asynchronous and infrequent, an average of eight times per 3.25-h trial in our experiments, its effect is minimal. The total energy is approximately 53 J, or 0.06% of the system energy usage shown in Figure 28.

5. RELATED WORK

PowerScope was the first tool that used profiling to measure energy usage. One alternative strategy [Bellosa 2000; Cignetti et al. 2000; Lorch 1995; Zeng et al. 2002] is to develop an energy model that quantifies power usage of specific system states. When an application executes, one estimates energy use by measuring time spent in each state and counting the number of state transitions. A drawback of this approach is that it is nontrivial to develop an energy model that accounts for all significant sources of power usage. Further, it is difficult to imagine expending this approach to capture the effects of proprietary, hardware-based power management—one example is Lucent 802.11 wireless network cards. A significant advantage of energy accounting is that it can separate out the energy consumption of asynchronous activities such as disk spin-up that PowerScope assigns to the currently executing process.

Another alternative for energy measurement is low-level power-analysis using instruction-level [Tiwari et al. 1994; Lee et al. 1997] or hardware [Brooks

et al. 2000; Vijaykrishnan et al. 2000] models. Because of their detailed nature, these models can provide accurate power estimates for small kernels of code. However, it is not clear that they can scale to large multithreaded applications such as Web browsers toward which PowerScope is targeted. CASTLE [Joseph and Martonosi 2001] uses a hybrid approach that combines a model of the microarchitecture with runtime analysis of processor performance counters to estimate CPU power consumption.

This work is also the first to demonstrate the energy benefits of application adaptation and the first to show that the operating system can effectively manage battery resources using feedback and application history. More recently, Ellis' EcoSystem [Zeng et al. 2002] project also explored operating system battery management. EcoSystem uses a hybrid measurement approach—it embeds an energy model in the operating system and accounts for inaccuracies with direct battery measurement similar to that used in Odyssey. EcoSystem meets target battery lifetimes by descheduling applications to leverage nonideal battery characteristics; that is, EcoSystem degrades performance rather than fidelity. EcoSystem can also provide isolation for energy resources by enforcing user-specified energy budgets for each application. In contrast, Odyssey uses a cooperative model: it allows each application to specify the range of adaptation it supports and does not enforce fidelity decisions. Neugebauer and McAuley [2001] describe how energy-awareness might be integrated into the Nemesis OS with an economic model for energy allocation. Yuan's GRACE-1 [Yuan et al. 2003] investigates how to tie together application adaptation and voltage scaling in a coordinated framework.

Chase's Muse system [Chase et al. 2001] reduces hosting center energy needs by managing server resources. Muse employs an economic model in which customers bid for service. When the marginal benefit of increased revenue exceeds the dynamic cost of energy usage, Muse powers up additional servers. While Muse's goal, maximizing service center profit, is very different from our goal of meeting desired battery lifetimes, both systems share the general approach of trading reduction of service for energy conservation.

At a broader scope, the large body of existing work in processor [Flautner et al. 2001; Lorch and Smith 2001; Weiser et al. 1994], storage [Douglis et al. 1995; Lu and De Micheli 1999], and network power management [Kravets and Krishnan 1998; Kravets et al. 1999] complements our work in energy-aware adaptation. As the results in Section 3 show, the benefits of energy-aware adaptation increase as hardware power management improves. Similarly, we expect ongoing work in compiler-based energy optimizations [Simunic et al. 1999; Tiwari et al. 1996] to prove synergistic with energy-aware adaptation.

6. CONCLUSION

Relentless pressure to make mobile computers lighter and more compact places severe restrictions on battery capacity. At the same time, mobile software continues to grow in complexity, hence increasing energy demand. Reconciling these opposing concerns by exploiting remote infrastructure is possible, but uses energy for wireless communication.

Energy-aware adaptation introduces flexibility into this overconstrained solution space. Rather than making static tradeoffs in hardware and software design, we defer these tradeoffs. At runtime, more accurate knowledge of energy supply and demand allows better decisions that resolve the tension between energy conservation and usability. Our results confirm that this approach yields substantial energy savings and can be effectively combined with existing hardware-centric approaches.

ACKNOWLEDGMENTS

We thank Dushyanth Narayanan for his assistance in incorporating his history-based prediction work into this project. We also thank Brian Noble, Kip Walker, and Eric Tilton for their contributions to the Odyssey platform. Keith Farkas, Todd Mowry, and Dan Siewiorek provided valuable insights throughout this project.

REFERENCES

ANDERSON, J. M., BERC, L. M., DEAN, J., GHEMAWAT, S., HENZINGER, M. R., LEUNG, S.-T. A., SITES, R. L., VANDEVOORDE, M. T., WALDSPURGER, C. A., AND WEIHL, W. E. 1997. Continuous profiling: Where have all the cycles gone? In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (SOSP, Saint-Malo, France). ACM Press, New York, NY, 1–14.

BARTLETT, J. F., BRAKMO, L. S., FARKAS, K. I., HAMBURGEN, W. R., MANN, T., VIREDAZ, M. A., WALKSPURER, C. A., AND WALLACH, D. A. 2000. *The Itsy Pocket Computer*. WRL Tech. note 2000.6. Compaq Western Research Laboratory, Palo Alto, CA.

BELLOSA, F. 2000. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop* (Kolding, Denmark). ACM Press, New York, NY.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture* (ISCA, Vancouver, Canada). 83–94.

CHASE, J. S., ANDERSON, D. C., THAKAR, P. N., VAHDAT, A. M., AND DOYLE, R. P. 2001. Manging energy and server resources in hosting clusters. In *Proceedings of the 18th Symposium on Operating Systems Principles* (SOSP, Banff, Canada). 103–116.

CIGNETTI, T. L., KOMAROV, K., AND ELLIS, C. S. 2000. Energy estimation tools for the Palm. In *Modeling, Analysis and Simulation of Wireless and Mobile Systems* (Boston, MA).

DALLAS SEMICONDUCTOR CORP. 1999. *DS2437 Smart Battery Monitor*. Dallas Semiconductor Corp., Dallas, TX.

DOUGLIS, F., KRISHNAN, P., AND BERSHAD, B. 1995. Adaptive disk spin-down policies for mobile computers. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing* (Ann Arbor, MI). 121–137.

FARKAS, K. I., FLINN, J., BACK, G., GRUNWALD, D., AND ANDERSON, J. 2000. Quantifying the energy consumption of a pocket computer and a Java virtual machine. In *Proceedings of ACM SIGMETRICS* (Santa Clara, CA). ACM Press, New York, NY.

FLAUTNER, K., REINHARDT, S., AND MUDGE, T. 2001. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking* (MOBICOM '01, Rome, Italy). 260–271.

FLINN, J., DE LARA, E., SATYANARAYANAN, M., WALLACH, D. S., AND ZWAENEPOEL, W. 2001. Reducing the energy usage of office applications. In *Proceedings of the FIP/ACM International Conference on Distributed Systems Platforms* (Middleware 2001, Heidelberg, Germany). ACM Press, New York, NY.

FLINN, J. AND SATYANARAYANAN, M. 1999a. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles* (SOSP, Kiawah Island, SC). ACM Press, New York, NY, 48–63.

FLINN, J. AND SATYANARAYANAN, M. 1999b. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications* (New Orleans, LA). IEEE Computer Science Press, Los Alamitos, CA, 2–10.

FOX, A., GRIBBLE, S. D., BREWER, E. A., AND AMIR, E. 1996. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of the Seventh International ACM Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS-VII, Cambridge, MA). ACM Press, New York, NY, 160–170.

HAMBURGEN, W. R., WALLACH, D. A., VIREDAZ, M. A., BRAKMO, L. S., WALDSPURGER, C. A., BARTLETT, J. F., MANN, T., AND FARKAS, K. I. 2001. Itsy: Stretching the bounds of mobile computing. *IEEE Computer 13*, 3 (April), 28–35.

INTEL CORPORATION AND MICROSOFT CORPORATION. 1996. *Advanced Power Management (APM) BIOS Interface Specification*. Intel Corporation, Santa Clara, CA, and Microsoft Corporation, Redmond, WA.

INTEL, MICROSOFT, AND TOSHIBA. 1998. *Advanced Configuration and Power Interface Specification*. Intel, Microsoft, and Toshiba. Available online at http://www.teleport.com/~acpi/.

JOSEPH, R. AND MARTONOSI, M. 2001. Run-time power estimation in high-performance microprocessors. In *Proceedings of the 2001 Symposium on Low Power Electronics and Design* (Huntington Beach, CA).

KLAIBER, A. 2000. The technology behind Crusoe processors. Tech. rep. (Jan.). Transmeta Corporation, Santa Clara, CA.

KRAVETS, R. AND KRISHNAN, P. 1998. Power management techniques for mobile communication. In *Proceedings of The Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking* (MOBICOM '98, Dallas, TX). ACM Press, New York, NY, 157–168.

KRAVETS, R., SCHWAN, K., AND CALVERT, K. 1999. Power-aware communication for mobile computers. In *Proceedings of The 6th International Workshop on Mobile Multimedia Communication* (San Diego, CA).

LEE, M. T.-C., TIWARI, V., MALIK, S., AND FUJITA, M. 1997. Power analysis and low-power scheduling techniques for embedded DSP software. *IEEE Trans. VLSI Syst. 5*, 1 (March), 123–135.

LORCH, J. R. 1995. A complete picture of the energy consumption of a portable computer. M.S. thesis, Department of Computer Science, University of California at Berkeley, Berkeley, CA.

LORCH, J. R. AND SMITH, A. J. 2001. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of ACM SIGMETRICS* (Cambridge, MA). ACM Press, New York, NY.

LU, Y.-H. AND DE MICHELI, G. 1999. Adaptive hard disk power management on personal computers. In *Proceedings of the 9th Great Lakes Symposium on VLSI* (Ypsilanti, MI). 50–53.

NARAYANAN, D., FLINN, J., AND SATYANARAYANAN, M. 2000. Using history to improve mobile application adaptation. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications* (Monterey, CA). IEEE Computer Society Press, Los Alamitos, CA, 30–41.

NEUGEBAUER, R. AND MCAULEY, D. 2001. Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In *Procceedings of the 8th Workshop on Hot Topics in Operating Systems* (HotOS-VIII, Schloss Elmau, Germany).

NOBLE, B. D., SATYANARAYANAN, M., NARAYANAN, D., TILTON, J. E., FLINN, J., AND WALKER, K. R. 1997. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (SOSP, Saint-Malo, France). ACM Press, New York, NY, 276–287.

OTHMAN, M. AND HAILES, S. 1998. Power conservation strategy for mobile computers using load sharing. *Mobile Comput. Commun. Rev. 2*, 1 (Jan.), 44–51.

RUDENKO, A., REIHER, P., POPEK, G. J., AND KUENNING, G. H. 1998. Saving portable computer battery power through remote process execution. *Mobile Comput. Commun. Rev. 2*, 1 (Jan.), 19–26.

SBS IMPLEMENTERS FORUM. 1998. *Smart Battery Data Specification, Revision 1.1*. SBS Implementers Forum. Available online at http://www.sbs-forum.org/.

SIMUNIC, T., BENINI, L., AND DE MICHELI, G. 1999. Energy-efficient design of battery-powered embedded systems. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design* (San Diego, CA). 212–217.

TIWARI, V., MALIK, S., AND WOLFE, A. 1994. Power analysis of embedded software: A first step towards software power minimization. *IEEE Trans. VLSI Syst. 2*, 4 (Dec.), 437–445.

TIWARI, V., MALIK, S., WOLFE, A., AND TIEN-CHIEN, L. M. 1996. Instruction level power analysis and optimization of software. *J. VLSI Signal Process. 13*, 2 (Aug.), 1–18.

USAR SYSTEMS, INC. 1999. *USAR ACPITroller II—Zero-Power ACPI KBC with Built-in Smart Battery System Manager*. USAR Systems, Inc., New York, NY.

VIJAYKRISHNAN, N., KANDEMIR, M., IRWIN, M. J., KIM, H. S., AND YE, W. 2000. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture* (ISCA, Vancouver, B. C., Canada). 95–106.

VIREDAZ, M. A. 1998. *The Itsy Pocket Computer Version 1.5 User's Manual*. WRL Technical Note TN-54. Compaq Western Research Laboratory.

WAIBEL, A. 1996. Interactive translation of conversational speech. *IEEE Comp. 29*, 7 (July), 41–48.

WEISER, M., WELCH, B., DEMERS, A., AND SHENKER, S. 1994. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Symposium on Operating System Design and Implementation* (OSDI, Monterey, CA). 13–23.

YUAN, W., NAHRSTADT, K., ADVE, S. V., JONES, D. L., AND KRAVETS, R. H. 2003. Design and evaluation of a cross-layer adaptation framework for mobile multimedia systems. In *Proceedings of Multimedia Computing and Networking* (Santa Clara, CA).

ZENG, H., ELLIS, C. S., LEBECK, A. R., AND VAHDAT, A. 2002. 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).

ZHANG, X., WANG, Z., GLOY, N., CHEN, J. B., AND SMITH, M. D. 1997. System support for automated profiling and optimization. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (SOSP, Saint-Malo, France). ACM Press, New York, NY.

Received October 2002; revised March 2003; accepted September 2003