Energy-aware adaptation for mobile applications

Jason Flinn and M. Satyanarayanan
School of Computer Science
Carnegie Mellon University

Abstract

In this paper, 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 show how applications can dynamically modify their behavior to conserve energy. We then show how the Linux operating system can guide such adaptation to yield a battery-life of desired duration. By monitoring energy supply and demand, it is able to select the correct tradeoff between energy conservation and application quality. Our evaluation shows that this approach can meet goals that extend battery life by as much as 30%.

1 Introduction

Energy is a vital resource for mobile computing. There is growing consensus that advances in battery technology and low-power circuit design cannot, by themselves, meet the energy needs of future mobile computers — the higher levels of the system must also be involved [1, 7].

In this paper, we explore how applications can dynamically modify their behavior to conserve energy. To guide such adaptation, the operating system monitors energy supply and demand. When energy is plentiful, application behavior is biased toward a good user experience; when it is scarce, the behavior is biased toward energy conservation.

To validate the energy benefits of adaptation, we present results from a detailed study of applications running on the Odyssey platform for mobile computing. Our results show energy reductions in the range of 7% to 72%, with a mean of 36%. Combined with hardware power management, we achieve overall reductions between 31% and 76%, with a mean of 50% — in effect, doubling battery life.

This research was supported by the National Science Foundation (NSF) under grant number CCR-9901696, and the Air Force Materiel Command (AFMC) under DARPA contract number F19628-96-C-0061. Addition support was provided by IBM. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, AFMC, DARPA, IBM, CMU, or the U.S. Government.

SOSP-17 12/1999 Kiawah Island, SC

Our measurements also suggest a novel approach to reducing the energy drain of the display, an important but difficult challenge. Using this approach, we project a further energy reduction ranging from 7% to 29%.

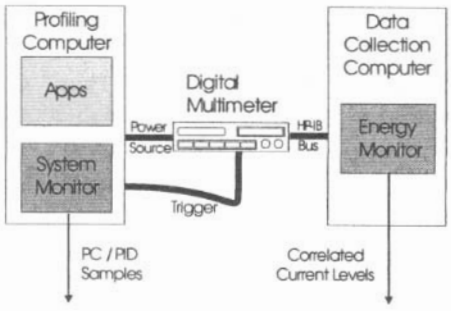
Finally, we show how the operating system can control adaptation by concurrent applications to give a battery life of user-specified duration. To perform this control, we have extended Odyssey to predict future energy demand from measurements of past usage. When there is substantial mismatch between predicted demand and available energy, Odyssey notifies applications to adapt. Using this approach, we demonstrate that Odyssey can extend battery-life to meet user-specified goals that vary by as much as 30%.

We begin with brief overviews of PowerScope, a tool we built to profile energy usage, and Odyssey. Three major sections follow: Section 3, on energy savings through adaptation; Section 4, on reducing display energy usage; and Section 5, on achieving a desired battery life. We close with a summary of related work and future plans.

2 Background

2.1 The PowerScope energy profiler

PowerScope is a tool for mapping energy consumption to specific software components. Its functionality and design are inspired by CPU profilers such as `prof` and `gprof` that help expose code components wasteful of processor cycles. Using PowerScope, one can determine what fraction of the total energy consumed during a certain time period is due to specific processes. Further, one can determine the energy



This hardware setup is used during PowerScope data collection. A data collection computer distinct from the profiling computer controls the multimeter and stores samples from it. Later, program counter and process id samples are correlated offline with current levels to yield energy profiles.

Figure 1. Data collection in PowerScope

Process	CPU Time(s)	Total Energy(J)	Average Power(W)
/usr/odyssey/bin/xanim	66.57	643.17	9.66
/usr/X11R6/bin/X	35.72	331.58	9.28
Kernel	50.89	328.71	6.46
Interrupts-WaveLAN	18.62	165.88	8.91
/usr/odyssey/bin/odyssey	12.19	123.40	10.12
Total	183.99	1592.75	8.66

Energy Usage Detail for process /usr/odyssey/bin/odyssey

Procedure	CPU Time(s)	Total Energy(J)	Average Power(W)
_Dispatcher	0.25	2.53	10.11
_IOMGR_CheckDescriptors	0.17	1.74	10.23
_sftp_DataArrived	0.16	1.68	10.48
_rpc2_RecvPacket	0.16	1.67	10.41
_ExaminePacket	0.16	1.66	10.35

This figure shows a sample energy profile. The first table summarizes the energy usage by process, while the table below shows a portion of the detailed profile for a single process. Only part of the full profile is shown.

Figure 2. Example of an energy profile

consumption of individual procedures within a process. By providing fine-grained feedback, PowerScope helps expose system components most responsible for energy consumption. Since PowerScope was recently described in detail [8], we only provide a brief overview here.

PowerScope uses statistical sampling to profile the energy usage of a computer system. To reduce overhead, profiles are generated in two stages. During the data collection stage, shown in Figure 1, the tool samples power consumption as well as the program counter (PC) and process identifier (PID) of the code executing on the profiling computer. A digital multimeter, currently a Hewlett Packard 3458a, samples the current drawn by the profiling computer through its external power input. Since the input voltage on computers is well-controlled (to within 0.25% in our measurements), current samples alone are adequate to infer energy consumption. The output of this stage consists of a sequence of current level samples and a correlated sequence of PC/PID samples. In a later off-line stage, PowerScope combines these sequences with symbol table information from binaries and shared libraries on the profiling computer. The result is an energy profile such as that shown in Figure 2.

2.2 The Odyssey platform for adaptation

The design rationale and architecture of Odyssey were presented in an earlier paper [17]. Adaptation in Odyssey involves the trading of data quality for resource consumption. For example, a client playing full-color video data from a server could switch to black and white video when bandwidth drops, rather than suffering lost frames. Similarly, a map application might fetch maps with less detail rather than suffering long transfer delays for full-quality maps.

Odyssey captures this notion of data degradation through an attribute called *fidelity*, that defines the degree to which data presented at a client matches the reference copy at a server. Fidelity is a type-specific notion since different kinds of data can be degraded differently. Since the minimal level of fidelity acceptable to the user can be both time and application dependent, Odyssey allows each application to specify the fidelity levels it currently supports.

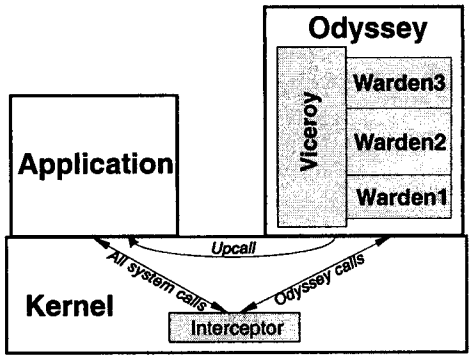


Figure 3. Odyssey architecture

Odyssey supports concurrent adaptation by diverse applications. The client architecture providing this support is shown in Figure 3. Odyssey is conceptually part of the operating system, even though it is implemented in user space for simplicity. The *viceroy* is the Odyssey component responsible for monitoring the availability of resources and managing their use. Code components called *wardens* encapsulate type-specific functionality. There is one warden for each data type in the system. We have built four adaptive applications on top of Odyssey: a video player, a speech recognizer, a map viewer, and a Web browser. Such multimedia and speech-enabled applications are of growing importance in mobile computing, although they are not yet as common as spreadsheets and word processors. Relevant details of these applications are presented later.

Odyssey is integrated into Linux as a new VFS file system, along with a set of API extensions for expressing resource expections. If resource levels stray beyond an application's expectation, Odyssey notifies it through an upcall. The application then adjusts its fidelity to match the new resource level, and communicates a new set of expectations to Odyssey. Some applications, such as our Web browser and map viewer, use a proxy to avoid modifications to application source code. Other applications, such as our video player and speech recognizer, are modified to interact directly with Odyssey.

The initial Odyssey prototype only supported network bandwidth adaptation. The work reported here extends Odyssey to support energy adaptation.

3 Energy impact of fidelity

Does lowering data fidelity yield significant energy savings? This was the crucial question facing us when we began this work. Incorporating support for energy-aware adaptation into Odyssey is futile if the potential savings are meager.

We were also keen to confirm that the energy savings from lowering fidelity could enhance those achievable through well-known hardware power management techniques such as turning off the disk or slowing the CPU [6, 13, 16]. Although these distinct approaches to energy savings seem composable, we wanted to verify this experimentally.

3.1 Methodology

To answer these questions, we measured the energy used by the Odyssey video player, speech recognizer, map viewer, and Web browser. We first observed the applications as they operated in isolation, and then as they operated concurrently. To maintain fidelity constant throughout an experiment, we disabled Odyssey's dynamic adaptation capability.

We explored sensitivity of energy consumption to data content by using four data objects for each application: that is, four video clips, four speech utterances, four maps, and four Web images. We first measured the baseline energy usage for each object at highest fidelity with hardware power management disabled. Next, we measured energy usage with hardware-only power management. Then, for each lower fidelity level we measured energy usage with hardware power management enabled.

This sequence of measurements is directly reflected in the format of the graphs presenting our results: Figures 6, 8, 10 and 13. Since a considerable amount of data is condensed into these 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 6. 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 total energy usage, and the shadings within each bar show energy usage by software component. The component labelled "Idle" aggregates samples that occurred while executing the kernel idle procedure — effectively a Pentium `hlt` instruction. The component labelled "WaveLAN" aggregates samples that occurred during network interrupts.

For each data set, the first and second bars, labelled "Baseline" and "Hardware-Only Power Mgmt.", show energy usage at full fidelity with and without hardware power management. Each of the remaining bars show the energy usage at a different 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 savings directly attributable to reduction in fidelity.

The measurements for the bars labelled "Hardware-Only Power Mgmt." were obtained by powering down as many hardware components as possible for each application. For example, we placed the disk in standby mode after 10 seconds of inactivity. Further, we modified the network communication package used by Odyssey to place 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. To ensure good experimental control, we disabled BIOS-level hardware power management. While these hardware power management techniques are simple, they combine to yeild up to a 34% reduction in energy usage.

Component	State	Power (W)
Display	Bright	4.54
	Dim	1.95
WaveLAN	Idle	1.46
	Standby	0.18
Disk	Idle	0.88
	Standby	0.24
Other	Idle	3.20

Background (display dim, WaveLAN & disk standby) = 5.6 W.

This figure shows the measured power consumption of components of the IBM 560X laptop. Power usage is slightly but consistently superlinear; for example, the laptop uses 10.28 W when the screen is brightest and the disk and network are idle — 0.21 W more than the sum of the individual power usage of each component. The last row shows the power used when the disk, screen, and network are all powered off. Each value is the mean of five trials — in all cases, the sample standard deviation is less than 0.01 W.

Figure 4. Power consumption of IBM ThinkPad 560X

3.2 Experimental setup

All measurements reported in this paper were obtained on a 233 MHz Pentium IBM ThinkPad 560X laptop with 64 MB of memory, running the Linux 2.2 operating system. This machine was configured as an Odyssey client and communicated with servers over a 2 Mb/s wireless WaveLAN network operating at 900 MHz. The servers were 200 MHz Pentium Pro desktop computers with 64 MB of memory.

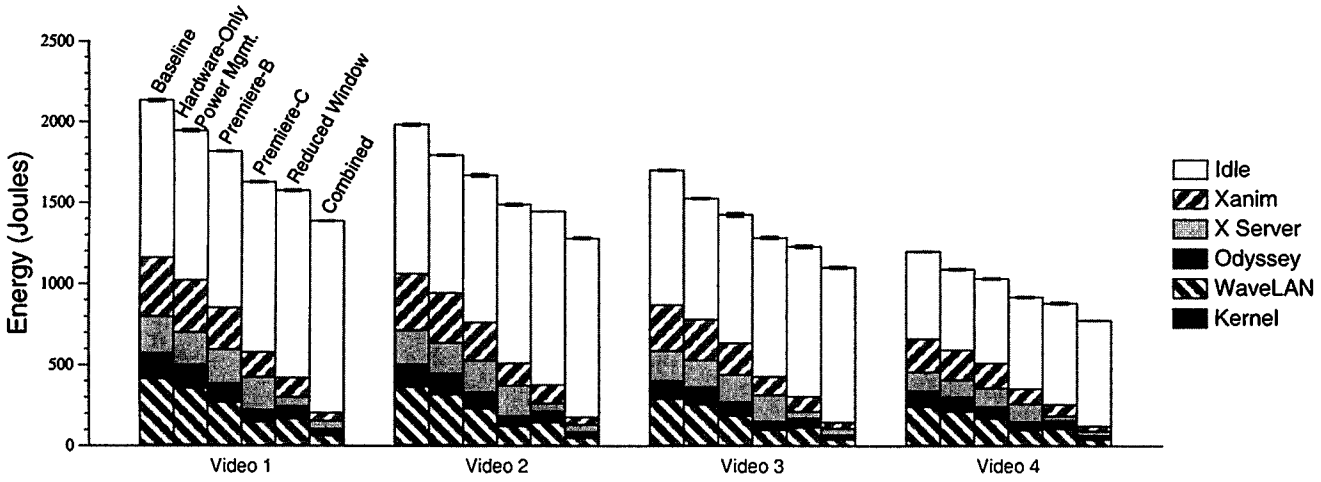
We profiled energy usage on the client with PowerScope, sampling approximately 600 times per second. To avoid confounding effects due to non-ideal battery behavior, the client used an external power supply. Further, to eliminate the effects of charging, the client's battery was removed.

Figure 4 shows the power usage of several hardware components of this laptop. To obtain these measurements, we used PowerScope to measure the change in power usage as we ran benchmarks which varied the power states of individual hardware components.

3.3 Video player

3.3.1 Description

We first measured the impact of fidelity on the video application shown in Figure 5. Xanim fetches videos from a server through Odyssey and displays them on the client. It supports two dimensions of fidelity: varying the amount of



This figure shows the energy used to display four QuickTime/Cinepak videos from 127 to 226 seconds in length, ordered from right to left above. For each video, the first bar shows energy usage without hardware power management or fidelity reduction. The second bar shows the impact of hardware power management alone. The next two show the impact of lossy compression. The fifth shows the impact of reducing the size of the display window. The final bar shows the combined effect of lossy compression and window size reduction. The shadings within each bar detail energy usage by software component. Each value is the mean of five trials — the error bars show 90% confidence intervals.

Figure 6. Energy impact of fidelity for video playing

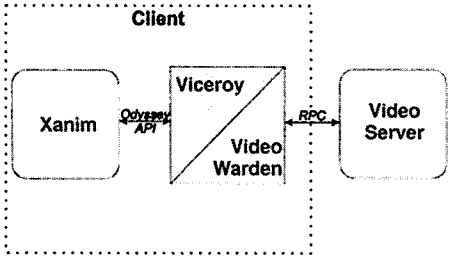


Figure 5. Odyssey video player

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 6 shows the energy used to display four videos at different fidelities. At 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. Most of the remaining energy is consumed by asynchronous network interrupts, the Xanim video player, and the X server.

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 disk remains in standby mode for the entire duration of an experiment.

The bars labelled Premiere-B and Premiere-C in Figure 6 show the impact of lossy compression. Premiere-C, at the highest compression, consumes 16–17% less energy than hardware-only power management. Note that these gains are understated due to the bandwidth limitation imposed by our wireless network. With a higher-bandwidth network, we could raise baseline fidelity and thus transmit better video quality when energy is plentiful. The relative energy savings of Premiere-C would then be higher.

By examining the shadings of each bar in Figure 6, we see 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 conjecture 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.

To validate this conjecture, we measured the effect of halving both the height and width of the display window, effectively introducing a new dimension of fidelity. As Figure 6 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 the techniques (hardware, Premiere-C, and reduced window) together yields about a 35% reduction.

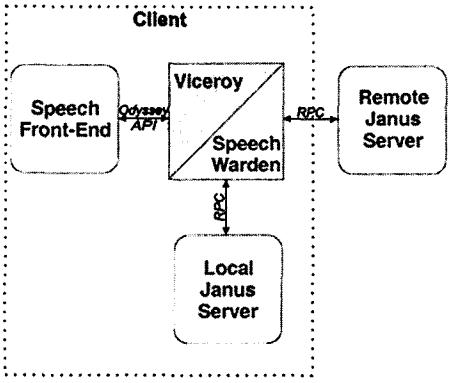


Figure 7. Odyssey speech recognizer

From the viewpoint of further energy reduction, the rightmost bar of each data set in Figure 6 offers a pessimistic message: there is little to be gained by further efforts to reduce fidelity. Virtually all the energy used at this fidelity is in the idle state, with the display consuming a large share. Turning off the display is, of course, not an option when watching video. We will return to this point in Section 4.

3.4 Speech recognizer

3.4.1 Description

Our second application is an adaptive speech recognizer. As shown in Figure 7, it consists of a front-end that generates a speech waveform from an utterance and submits it via Odyssey to a local or remote instance of the Janus speech recognition system [23].

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 five reduction in data volume with little computational overhead.

Fidelity is lowered in this application by using a reduced vocabulary and a less complex acoustic model. This substantially reduces the memory footprint and processing required for recognition, 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 in the reduced vocabulary. This helps counterbalance the effects of reducing the sophistication of the acoustic model.

3.4.2 Results

Figure 8 presents our measurements of client energy usage when recognizing four pre-recorded utterances using local, remote and hybrid strategies at high and low fidelity. 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.

Hardware-only 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. More complex recognition tasks may trigger disk activity and hence show less benefit from hardware power management.

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

Remote recognition at full fidelity reduces energy usage by 33–44% below that obtained by using hardware-only power management. If fidelity is also reduced, the corresponding savings is 42–65%. These figures are comparable to the energy savings for remote execution reported in the literature for other compute-intensive tasks [18]. As the shadings in the fourth and fifth bars of each data set in Figure 8 indicate, 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, thus shortening this interval and yielding additional energy savings.

Hybrid recognition offers slightly greater energy savings than remote recognition: 47–55% at full fidelity, and 53–70% at low fidelity, both relative to hardware-only power management. Hybrid recognition increases the fraction of energy used by the local Janus code; but this is more than offset by the reduction in energy for 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 the baseline. In practice, the optimal strategy will depend on resource availability and the user's tolerance for low-fidelity recognition.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.