Energy Reduction by Workload Adaptation in a Multi-Process Environment [*]

Changjiu Xian
Department of Computer Science
Purdue University
West Lafayette, IN 47907
cjx@cs.purdue.edu

Yung-Hsiang Lu
School of Electrical and Computer Engineering,
Purdue University
West Lafayette, IN 47907
yunglu@purdue.edu

Abstract

Reducing energy consumption is an important issue in modern computers. Dynamic power management (DPM) has been extensively studied in recent years. One approach for DPM is to adjust workloads, such as clustering or eliminating requests, as a way to trade-off energy consumption and quality of services. Previous studies focus on single processes. However, when multiple concurrently running processes are considered, workload adjustment must be determined based on the interleaving of the processes' requests. When multiple processes share the same hardware component, adjusting one process may not save energy. This paper presents an approach to assign energy responsibility to individual processes based on how they affect power management. The assignment is used to estimate potential energy reduction by adjusting the processes. We use the estimation to guide runtime adaptation of workload behavior. Experiments demonstrate that our approach can save more energy and improve energy efficiency.

1. Introduction

Energy reduction is an important design issue in computer systems (special issue in IEEE Computer December 2003). Many techniques [1] have been proposed in the last several years. Among these techniques, dynamic power management (DPM) has been widely studied. DPM saves energy by shutting down hardware components when they are idle. Since shutting down and waking up a component consume energy, only long idle periods can justify such overhead and obtain energy savings. Many studies focus on improving the power manager so that it can predict future idleness accurately. While improving the power manager can effectively reduce the energy during long idle periods, a workload without long idle periods provides no opportunity for the power manager to save energy. To resolve this, some studies adjust the workload to create long idle periods. These studies can be classified into two major categories: (a) clustering (also called "rescheduling") programs' requests [2, 7, 9, 10] and (b) removing requests [5, 6, 11]. These workload techniques often need to trade off performance and quality of service or use extra resources such as memory buffers. Since unused memory banks can be put into a low-power state [4], using memory for buffering trades off such opportunities. Thus, the workload techniques should only be applied when substantial energy can be saved.

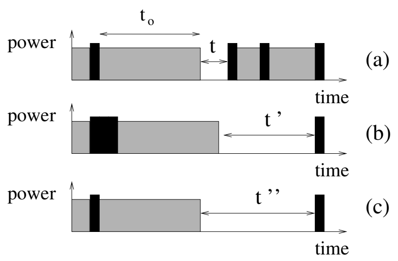

Figure 1. DPM and workload adjustment: (a) shutdown with a timeout, (b) clustering requests, and (c) removing requests.

A fundamental question has not been fully addressed: how much energy can possibly be saved by adjusting the workload? We use Figure 1 to illustrate the concept. In the figure, each black bar indicates a request (e.g., a network packet to send or a disk read command). Each gray rectangle means the component (e.g. a network card or a disk) is idle but ready to serve requests. The component consumes energy during idleness. The component can be shut down and consumes no energy. Figure 1(a) shows the original requests and a power manager that shuts down the component after being idle for t_o. In Figure 1(b), the requests are clustered so the component can remain shut down (also called "sleeping") longer ($t' > t$) and save more energy. In Fig-

* This work is supported in part by National Science Foundation Career CNS-0347466 and Purdue Research Foundation. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the sponsors.

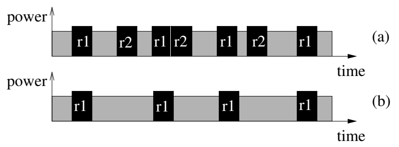

Figure 2. (a) The original requests. (b) Only process 2's requests are removed.

ure 1(c), two requests are removed and the component can sleep even longer ($t'' > t'$).

Figure 2 extends the concept to two concurrent processes, P_1 and P_2. In this figure, r_1 and r_2 represent the requests from P_1 and P_2, respectively. If all requests from P_2 are removed, as shown in Figure 2(b), the idle periods are still short because of P_1's requests. This example suggests that the energy reduction by modifying one process is affected by other processes.

This paper focuses on answering the following two questions: (a) If the requests from a single process are modified by either clustering or removing, how much energy can be saved? (b) When a new process executes or an existing process terminates, how should the other processes respond to obtain more energy savings and better energy efficiency? The efficiency is defined as the ratio of the amount of work to the energy consumption. We assign energy responsibility to individual processes and use the assignments to estimate the potential energy reduction by adjusting each process. The estimation of energy reduction is crucial to run-time adaptation because sufficient energy must be saved to justify the degraded performance. We implement our method in Linux and demonstrate substantial improvements in energy savings and efficiency by providing the estimation to application programs.

2. Related Work

Some studies have considered adjusting workloads for power management. Lu et al. [7] order and cluster the tasks for multiple devices to create long idle periods for power management. Weissel et al. [10] propose to assign timeouts to file operations so they can be clustered within the time constraints. Rong et al. [9] divide power management into system and component levels and propose clustering requests by modeling them as stochastic processes. Cai et al. [2] use buffers to cluster accesses to a device. Flinn et al. [5] reduce the quality of service such as the frame size and resolution of video when battery energy is scarce. Zeng et al. [11] assign an energy budget to processes and a process is not allowed to run when its budget is consumed. In a multi-process environment, the energy savings from adjusting one process can be affected by other processes. Our study provides guidence to the adjustment by estimating the potential energy savings when other concurrently running processes are considered.

We estimate the potential energy savings due to workload adjustment based on the processes' energy responsibilities. PowerScope [6] uses a multimeter to measure the whole computer's power consumption and correlates the measurements to software programs by sampling the program counter. Their study provides information about procedural level energy consumption. Chang et al. [3] conduct a similar measurement with a special hardware called Energy Counter, which reports when a predefined amount of energy is consumed. ECOSystem [11] models the energy consumption of different components individually and assigns energy to the processes by monitoring their usage of individual components. Their study controls processes' energy consumption using operating system (OS) resource allocation. Neugebauer et al. [8] perform similar energy assignment in a system called Nemesis OS providing QoS guarantees. None of these studies examine the relationship between processes' energy responsibilities and the potential energy savings from adjusting the processes. Moreover, they do not examine how energy sharing in a multi-process environment influences the effectiveness of workload adjustment. Hence, these studies are insufficient for estimating the energy savings of workload adaptation in a multi-process environment.

3. Estimating Energy Savings

This section analyzes how energy is consumed in a multi-process environment and estimates the energy saving opportunities from adjusting the workload. We divide energy responsibility between the power manager and the user processes so the processes' energy assignments are independent of specific power management policies. We also divide energy responsibility among the processes based on how they affect the effectiveness of dynamic power management. The assignments are then used to estimate potential energy savings from changing workload. For simplicity, we first assume three power states: *busy* (serving requests from processes), *sleeping* (requests have to wait for the component to wake up), and *idle* (not serving requests but ready to serve without delay). We extend our method to multiple sleeping states in Section 3.3.

3.1. Energy Responsibility and Sharing

To determine the opportunity of energy reduction from improving the power manager or adjusting processes, we assign the energy consumption that can be reduced by improving the shutdown accuracy to the power manager and assign the other energy to the user processes. The energy assigned to a process can be reduced by adjusting the process. For example, if the component serves only a single request as shown in Figure 3 (a), the necessary energy consumption includes the wakeup energy (e_w), service energy (e_a), and the shutdown energy (e_d). Any additional energy

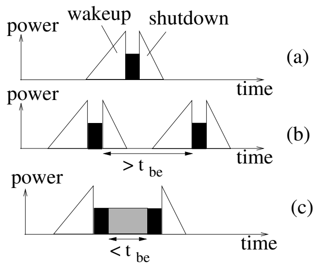

Figure 3. Energy responsibility of a process.

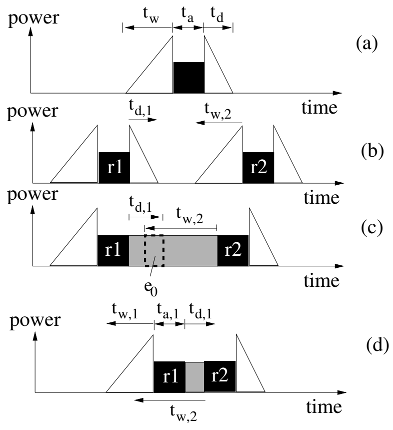

Figure 4. Two requests r_1 and r_2 from two processes. (a) Sharing periods. (b) No sharing between r_1 and r_2. (c) r_1 and r_2 share the energy e_o. (d) r_2 shares e_w with r_1's e_a and e_w.

can be reduced by performing wakeup and shutdown immediately before and after the service. The energy $e_w+e_a+e_d$ is responsible by the process because it can be reduced only by removing the request. When multiple requests access a component, we can calculate the necessary energy consumption based on the component's break-even time. The break-even time (t_{be}) is the minimum duration of an idle period during which the energy saved in the sleeping state can compensate the state-change energy (e_d+e_w) [1]. If the idle period is longer than t_{be} as shown in Figure 3 (b), the processes are responsible for only e_d+e_w in the idle period. If the idle period is shorter than t_{be} as shown in Figure 3 (c), the energy in the idle period cannot be reduced by shutdown and such idle energy is assigned to the processes.

We next divide the responsibility among processes to estimate the opportunity for energy reduction from adjusting each process. Suppose the two requests in Figure 3 (b) and (c) are from two different processes. In Figure 3 (b), each process is responsible for $e_w+e_a+e_d$ of its request. In Figure 3 (c), the energy consumption of the two requests are not separated by shutdown because the idle period is less than t_{be}. The processes share energy in this case. To calculate sharing for determining energy responsibility, we first extend the concept from Figure 3 (a). The process is responsible for energy e_w before a request and e_d after the request. The energy corresponds to time t_w and t_d as shown in Figure 4 (a); they are called *backward sharing period* and *forward sharing period*, respectively. Their values are defined as $\int_0^{t_w}\rho(t)dt = e_w$ and $\int_0^{t_d}\rho(t)dt = e_d$, here $\rho(t)$ is the power at time t. The value of t_w is calculated backward from the service, i.e., at the moment of the service, $t=0$. In the case of Figure 4 (a), t_w and t_d are just the component's wakeup delay (τ_w) and shutdown delay (τ_d), respectively. In the case of 4 (c), $\rho(t)=p_l$ (here p_l is the idle power) for calculating $t_{d,1}$ for r_1 and $t_{w,2}$ for r_2, so $t_{d,1}=e_d/p_l$ and $t_{w,2}=e_w/p_l$.

Two processes do not share energy if their t_w and t_d do not overlap, as shown in Figure 4 (b). When the idleness between the two requests becomes shorter as shown in Figure 4 (c), $t_{d,1}$ and $t_{w,2}$ may overlap and the two processes share energy. The energy during the period $t_{d,1}$ is $e_{d,1}$ and the energy during the period $t_{w,2}$ is $e_{w,2}$. The energy consumption in the overlapped period is e_o. If r_1 is removed and no longer responsible for any energy, e_o cannot be reduced by any power manager because r_2 needs energy $e_{w,2}$ to wake up the component and $e_{w,2}$ includes e_o. Similarly, removing r_2 cannot reduce e_o because r_1 needs energy $e_{d,1}$ to shut down the component. To reduce the shared energy e_o, both requests have to be removed. Therefore, both processes are responsible for the overlapped energy e_o. As the idle period becomes even shorter, $t_{w,2}$ can overlap with $t_{a,1}$ and even $t_{w,1}$, as shown in Figure 4 (d). The rationale of energy sharing is the same — the energy in the overlapped period can be reduced only by removing both processes. We can extend this approach to three or more processes by calculating their sharing periods. If their periods overlap, all processes associated with the overlapping equally share the energy during the overlapped interval.

3.2. Potential Energy Savings

Shared energy can be reduced only by adjusting all the sharing processes. Consequently, the potential energy savings from removing only one process' requests are $E_r = e_p - e_h$, here e_p is the process' responsible energy and e_h is the portion of the process' responsible energy shared with other processes. Clustering a process' requests all together is equivalent to two steps — removing all the process' requests first and then add the cluster back. The cluster consumes energy $e_w + \sum e_a + e_d$, here $\sum e_a$ is the sum of the service energy (e_a) of all the clustered requests. Hence, the potential energy savings from clustering a process' requests are $E_c = E_r - (e_w + \sum e_a + e_d)$. The energy assigned to the power manager is the potential savings to improve the shutdown accuracy. As a supplement to these potential energy savings, we also calculate the current energy savings E_s, namely, the energy savings that has been ob-

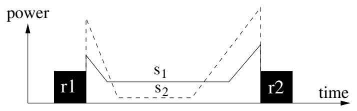

Figure 5. The power consumption of a component with two sleeping states.

tained. This is equal to the difference between the energy consumption without shutdown and the actual energy consumption. For example, for the idle period between the two requests in Figure 3 (b), the consumption without shutdown is $p_l \times t_l$ (here p_l is the idle power and t_l is the length of the idle period) and the actual consumption is $e_d + e_w$. The current energy savings are $E_s = p_l \times t_l - (e_d + e_w)$.

3.3. Multiple Sleeping States

As processes are only responsible for necessary (or minimal) energy consumption during idle periods, we extend the calculation of necessary consumption to components with multiple sleeping states. Let $s_1, s_2, ..., s_n$ be n sleeping states ordered by decreasing power consumption and increasing state-change delay ($\tau_d + \tau_w$) and energy ($e_d + e_w$). Each sleeping state has a corresponding break-even time from the idle state. When the idle period is longer than several break-even times, we need to determine which sleeping state to choose for minimum energy consumption. This state is not necessarily the lower-power sleeping state. Figure 5 is used to illustrate this concept. State s_2 consumes less power but has larger wakeup and shutdown energy than s_1. We can calculate the minimum length of an idle period when entering s_2 saves more energy. Let t be the length of an idle period; e_{s_1} and e_{s_2} are the state-change energy, t_{s_1} and t_{s_2} are state-change delay, and p_{s_1} and p_{s_2} are the power. Using the two states achieves the same energy savings if $e_{s_1} + p_{s_1}(t - t_{s_1}) = e_{s_2} + p_{s_2}(t - t_{s_2})$, or $t = \frac{e_{s_2} - e_{s_1} + p_{s_1} t_{s_1} - p_{s_2} t_{s_2}}{p_{s_1} - p_{s_2}}$. This is the minimum duration of an idle period to use s_2. Notice that this is different from s_2's break-even time because t_{be} is defined between a sleeping state and the idle state, not between two sleeping states. To calculate the sharing periods t_w and t_d for each request, we use e_w and e_d of the deepest sleeping state. This is because if there is only a single request, the component should be kept in the deepest sleeping state before and after serving the request in order to consume the minimum energy.

3.4. Runtime Adaptation

Previous sections show that, after assigning each process its energy responsibility, we can estimate the potential energy savings from adjusting the workload. In this section, we show that the estimation can be peformed at runtime such that processes can perform runtime adaptation to save energy, improve energy efficiency, or both. Specifically, we periodically calculate and assign energy responsibility to each process and inform the process of the potential energy savings E_c or E_r as defined in Section 3.2. We assume that the E_c and E_r calculated from the previous period can be used to estimate the following period. Processes can combine the estimation and their specific constraints (e.g., performance requirement) to determine when to perform clustering or removing.

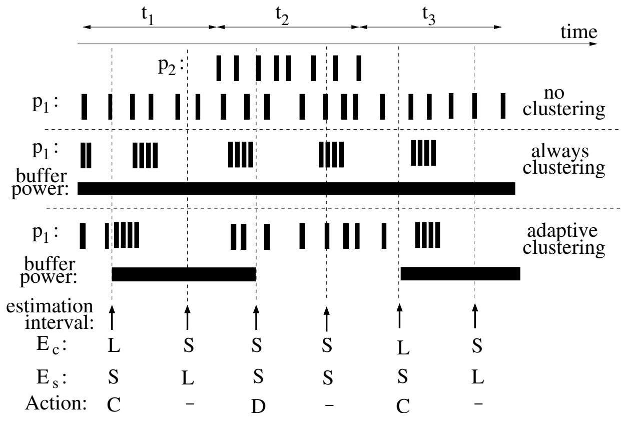

Figure 6. Comparison of clustering schemes when two processes (P_1 and P_2) running concurrently using the same hardware component. E_c: potential energy saving by clustering. E_s: current energy saving. L/S: large/small value. C/D: allocating/deallocating the buffer for clustering.

We use two techniques as examples of run-time adaptation: (a) Allocating a chunk of memory buffer to cluster a process' requests to a component. For example, additional memory is allocated for an audio/video streaming program to prefetch more frames from the server and the network card is used to fetch frames only when the buffered frames have been consumed. (b) Removing a process' requests by removing or suspending the process. For example, when battery energy is scarce, the user may be willing to terminate or suspend a low-priority program as a tradeoff for energy savings.

Figure 6 illustrates clustering guided by our estimation of potential energy savings. Processes P_1 and P_2 generate requests for the same hardware component (e.g., a network card). During t_1 and t_3, only P_1 is running. During t_2, P_2 is running concurrently with P_1. We assume P_2 does not allow clustering due to real-time constraints, such as a telephony program transmitting real-time voice data. When only P_1 is running, clustering P_1's requests can create long idle periods. When P_2 is also running, clustering P_1 cannot create long idle periods because P_2's requests are scattered.

The first two rows in Figure 6 show that no clustering technique is used. This case consumes no extra buffer power but misses the opportunity to save energy in t_1 and t_3 by clustering P_1's requests. The third and fourth rows cluster P_1's requests using buffer throughout the whole duration. Even though energy is saved for the component during t_1 and t_3, it does not save energy during t_2 while additional energy is consumed for the buffer memory. Alternatively, P_1 can perform adaptive clustering with the information of the potential energy savings (E_c) and the current energy savings (E_s), as calculated in Section 3.2. The value of E_c suggests whether it is beneficial to further cluster the requests but does not indicate whether the current running clustering saves energy. In contrast, E_s tells how much energy savings we have obtained but does not suggests the potential of further clustering. Consequently, if E_c is large, P_1 should cluster no matter whatever E_s is. If buffer has been allocated and both E_c and E_s are small, P_1 should stop clustering and release the buffer. This is because the small E_c suggests no potential for allocating more buffer for further clustering and the small E_s suggests that the current clustering is not beneficial. After the buffer is released, the unused memory banks can enter low-power state.

At the beginning of t_1, the potential savings are large (shown as "L") and the current savings are small (shown as "S") as shown in the figure. Specifically, the "large" or "small" are determined by comparing with the per-period energy consumption of the buffer memory allocated for clustering. As a result, P_1 clusters the requests. After clustering, the potential savings become small and the current savings become large. When P_2 starts running at the beginning of t_2, both E_c and E_s are small. This indicates that clustering is no longer beneficial so P_1 stops clustering and releases the buffer. When P_2 terminates at the end of t_2, the potential savings become large again and P_1 should cluster requests during t_3.

The method of using the estimation to guide runtime workload reduction is similar. Suppose a user would suspend a user process only if the suspension saves substantial energy (e.g., $> 10\%$). We can perform runtime adaptation using the estimation E_r and the current energy saving E_s. When E_r is large, we suspend the process to save energy. If the process is being suspended and E_s is small, the process is resumed to perform more service. Compared to simply terminating the process, this approach can save about the same amount of energy and perform more service. Hence, the energy efficiency is improved.

4. Experiments

4.1. Experimental Setup

Our prototype uses Integrated Development Platform (IDP) by Accelent Systems running Linux 2.4.18. The IDP uses Intel PXA250 as the processor and provides probing points to measure the power consumption of different hardware components. We install an Orinoco wireless network card and an IBM Microdrive. Table 1 shows the measured parameters of the components used in our experiments. We implemented a Linux kernel module to estimate energy savings. The timing information (e.g., the starting times and the ending times) of the processes' accesses to the devices are collected by modifying the device drivers. The workload consists of six programs: `madplay`: an audio player, `xmms`: an audio streaming program, `mpegplayer`: an MPEG video player, `gzip`: a compression tool, `scp`: a secure file transferring utility, `httperf`: a program retrieving web pages. These programs have different workload on different components. In our experiments, `scp` is always running as the background process to upload data files from the Microdrive to a remote server through the wireless network card. We choose `scp` because it has no stringent timing constraints (like an audio player). The other programs are occasionally selected to execute concurrently with `scp`. We use the degree of concurrency to indicate how many concurrent user processes are running. When the degree of concurrency is one, only `scp` is running. When the degree of concurrency is higher, the other six programs are randomly selected to execute. For example, when the degree of concurrency is three, two other programs execute concurrently with `scp`. We divide the whole duration of an experment into 300-second intervals and randomly determine a degree of concurrency for each interval. For both clustering and removal, we conduct five experiments with increasing average and maximum degree of concurrency. We perform removal or suspension for `scp` on the wireless card and clustering for `scp` on the Microdrive. For clustering, we allocate a memory buffer of 10MB to prefetch data from the Microdrive. The memory consumes 5×10^{-5} W for every page of 4KB. The power is calculated using the SDRAM datasheet from the Micron website. This is based on the assumption that unused memory can be turned off to save power [4]. A 0.65s timeout is used to shutdown Microdrive and a 0.15s timeout for the wirelsess card.

	meaning	microdrive	wireless
p_a (W)	active power	0.60	1.3tx,0.8rx
p_l (W)	idle power	0.59	0.75
p_s (W)	sleeping power	0.24/0.066	0.08
τ_d (s)	wakeup delay	0.159/0.160	0.03
τ_w (s)	shutdown delay	0.273/0.716	0.06
e_w (J)	wakeup energy	0.207/0.475	0.079
e_d (J)	shutdown energy	0.124/0.135	0.038
t_{be} (s)	break-even time	0.65/1.05	0.15

Table 1. Power parameters. The microdrive has two sleeping states, shown as "s_1/s_2".

4.2. Energy Savings and Efficiency

Figure 7 shows our experimental results. Figures 7 (a) and (c) show the energy savings of using clustering and removal for different degrees of concurrency. Figures 7 (b)

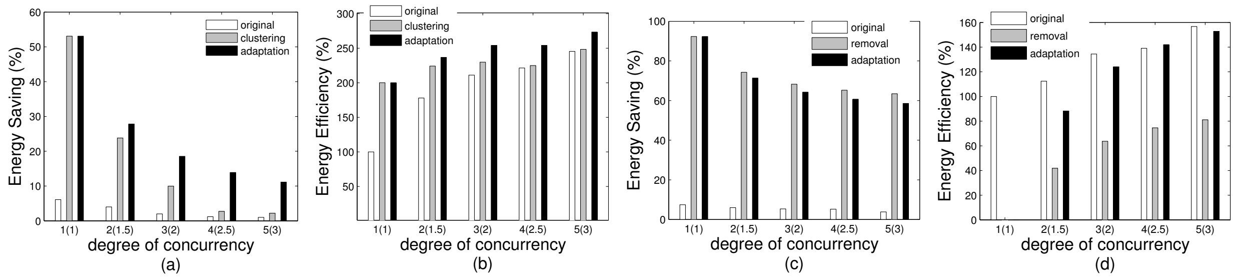

Figure 7. Energy savings and efficiency of clustering and removal for different degrees of concurrency. Degree x(y) means the maximum(average) degree of concurrency. Higher bars are better.

and (d) show the energy efficiency measured as the number of bytes processed by all programs for every Joule of energy. Figures 7 (b) and (d) are normalized to the original workload as 100%. Figure 7 (a) shows that clustering can save more energy (47%) than the original no clustering case when only `scp` is running and the degree of concurrency is one. As the degree increases, less energy can be saved by clustering because the concurrent processes create scattered requests. If `scp` continues clustering, little energy can be saved by the Microdrive and the network card. Meanwhile, energy is consumed by the buffer memory. In contrast, our adaptive method informs `scp` to stop clustering and release the buffer memory. Thus, our approach can save more energy and improve energy efficiency. Figures 7 (c) and (d) consider request removal. When the degree of concurrency is one and the requests are removed, over 92% energy can be saved as shown in Figure 7 (c). However, the efficiency is zero as shown in Figure 7 (d) because no data are copied by `scp`. As the degree of concurrency increases, `scp` can adaptively execute when the microdrive and the network card are in the active state after serving the other programs. Even though the amount of energy saved by adaptation is less than removing `scp` by up to 5% as shown in Figure 7 (c), the energy efficiency are significantly higher shown in Figure 7 (d). This is because we resume `scp` when the requests from other process are scattered on the network card so the energy during the scattered idleness is utilized to serve `scp`'s requests. The original method (no-removing) has high efficiency but it saves much less energy. The experimental results show the importance of considering concurrent processes to achieve better energy savings and efficiency.

5. Conclusion

This paper presents a method to save energy in an environment of multiple processes. We estimate how much energy can be saved when a process clusters or removes requests. By considering the energy sharing among multiple processes, a process can save more energy when it is responsible for most of the energy consumption. If a process is responsible for only a small portion of the energy, the process can stop clustering or removal to improve the energy efficiency. Experimental results indicate that energy savings and efficiency can be significantly affected by the presence of other concurrent processes.

References

[1] L. Benini and G. D. Micheli. System-Level Power Optimization: Techniques and Tools. *ACM TODAES*, April 2000.

[2] L. Cai and Y.-H. Lu. Dynamic Power Management Using Data Buffers. In *DATE*, pages 526–531, 2004.

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

[4] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Hardware and Software Techniques for Controlling DRAM Power Modes. *IEEE Transactions on Computers*, 50(11):1154–1173, November 2001.

[5] J. Flinn and M. Satyanarayanan. Energy-Aware Adaptation for Mobile Applications. In *ACM SOSP*, 1999.

[6] J. Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–10, 1999.

[7] Y.-H. Lu, L. Benini, and G. D. Micheli. Low-Power Task Scheduling for Multiple Devices. In *International Workshop on Hardware/Software Codesign*, pages 39–43, 2000.

[8] R. Neugebauer and D. McAuley. Energy is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS. In *Workshop on HotOS*, pages 59–64, 2001.

[9] P. Rong and M. Pedram. Hierarchical Power Management with Application to Scheduling. In *ISPLED*, pages 269–274, 2005.

[10] A. Weissel, B. Beutel, and F. Bellosa. Cooperative IO- A Novel IO Semantics for Energy-Aware Applications. In *OSDI*, pages 117–129, 2002.

[11] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing Energy As A First Class Operating System Resource. In *ASPLOS*, pages 123–132, 2002.