Fine-Grained Power Management Using Process-level Profiling

Hui Chen, Youhuizi Li and Weisong Shi
Department of Computer Science
Wayne State University
{huichen,huizi,weisong}@wayne.edu

***Abstract*—Low-power hardware design itself is not enough to solve the power problem of computer systems. Operating system level power saving strategies have been proved as effective complement to hardware methods. However, most of these strategies work on system level, and some of them may severely influence performance. In order to balance between performance and energy consumption, fine-grained methods, such as process-level power management, were proposed. These methods usually require realtime power information to make critical power-saving decisions, but most power profiling tools only supply component level power information. In this paper, we first propose a process-level power profiling tool called pTopW, which runs on Windows platform. pTopW supplies a group of APIs for power-aware system modules to acquire realtime power profiles and make power-aware decisions based on these information. In addition, we introduce a power-aware system module called EnergyGuard, which can eliminate energy wasted by abnormal-behavior applications. Through experiments, we found that the energy model we proposed have very good responsiveness. In addition, EnergyGuard is helpful to distinguish applications' abnormal power behaviors.**

***Keywords*-Power Profiling; EnergyGuard; pTopW;**

I. Introduction

In the past several years, power dissipation and subsequent problems have acquired comprehensive concerns. In order to solve these problems, we have totally changed the way of designing computer systems. One of the rules is that power became one of the first-class architectural design constraints, and performance was not the only consideration during the architecture design cycle [1]. Limiting power dissipation is not only important for desktop and mobile devices, such as laptop and smart phone, but also critical for high-end systems. For mobile devices, power dissipation is closely related to battery lifetime, which is one of the significant aspects of the user experience, because the energy density of current battery is limited. In addition, accompanied with the rapid expansion of Internet services, a lot of data centers were built to supply powerful and stable services for users. However, it is well known that a data center is power hungry and has very low energy efficiency because it's designed for peak workloads. To evaluate energy efficiency, the Green Grid group proposed the definition of power usage effectiveness (PUE) [2]. Power and the concomitant thermal issues significantly restrain the improvement of performance. For example, if the clock frequency of processor continues to increase, it may be inevitable to economically cool down the chips.

To solve the power problem and improve the energy efficiency, a lot of research have been done from different angles, such as low-power circuits design, power-aware hardware architecture and power-aware operating system design. The efforts in the early age [3] are mainly done on low-power circuits design. To do the power/performance evaluation, tools, such as Wattch [4] and SimplePower [5], were introduced to estimate the circuit level power. Even though these specialized low-power hardware design techniques have been proved as effective for power reduction, these techniques alone are not enough, and higher levels of power saving mechanisms becoming more and more important [4]. Vahdat *et al.* also claimed that both low-level mechanisms and higher-level policies are required for maximizing energy efficiency, and that all aspects of traditional maimizing performance aimed operating system design should be revisited [6]. Zeng *et al.* implemented ECOSystem [7], which, to our knowledge, is the first operating system which have seen power as the first-class resource. They tried to extend the battery lifetime by managing the system-level power dissipation.

Although current power management strategies that have been done on operating system are effective during a long period, the space for power saving decreased sharply with the development of computer hardware. A group of experiments done by Le Sueur and Heiser demonstrated that using DVFS may cause more energy consumption compared to the case of not using it [8]. The main reason which cause this situation is due to these power saving strategies which decrease the performance and thus require longer time to finished the task. On the other hand, static power have already dominated the system power dissipation.

To solve this situation, higher level power management techniques, which try to eliminate the energy waste done by tasks, applications, processes and threads, are proposed. However, these fine-grained power management strategies require realtime power information. Different with circuit level power estimation, which is used during the hardware design cycle, realtime power information is used on current platforms. Usually, there are two methods to acquire the realtime power information: direct measurement and software-based power profiling. The power measured directly with different kinds of meters cannot serve as realtime services because these meters are costly, and thus they are not suitable for normal users. Thus, this method can only be used during research even though it is accurate. High-end systems usually integrate

power sensors into the hardware to supply realtime power information. Integrating sensors into hardware, however, is also expensive, and the current usage is only limited to servers. Software-based power profiling is much more flexible and low-cost. Also, it can be used to estimate more fine-grained power information, which is not only useful for system to make power-efficient decisions, but also can be used to make software power/performance evaluations.

Most previous publications of this category [9], [10], [11] use a group of metrics that can reflect the power dissipation of the hardware to build the power model. These previous works can estimate the power accurately. However, these articles targeted component-level power estimation. Energy efficient strategies usually require process or thread level power profiles in able to manage energy as a kind of resource. For example, Cinder [12] operating system requires these information to allocate a "Reserve" for each application. In addition, some methods are too complicated to be used in realtime, otherwise, the overhead is too high. To limit the overhead, simplicity is also a critical constraint for power model. In our previous work [13], we had designed a power profiling tool for Linux platform. Our new work not only improves the energy model, but also evaluates the models in more details. In addition, we made several modifications to the architecture of pTop; this makes it easier to use on windows platform.

With the power profiles supplied by pTopW, we designed a fine-grained power-aware strategy called EnergyGuard. Unlike other methods, which try to decrease energy consumption by losing parts of the performance, EnergyGuard try to distinguish applications that have abnormal power behaviors and then take actions to eliminate the energy wasted by these processes. More specifically, we monitor the power dissipation of all processes, build the power history and detect abnormal process, whose power is always increased sharply and influence other processes. Then, we can inform users and avoid energy waste. Besides monitoring the safe processes, EnergyGuard can detect energy greedy malware in a relatively short time. Current EnergyGuard module mainly considers how to distinguish these abnormal processes and can only supply a simple power-saving strategy. How to design different strategies to control the power dissipation of these processes is part of our future work.

In this paper we have three contributions:

- We propose four energy models to estimate the energy consumption and the power of four main components. The experiment results show that these power model show good responsiveness most of the time. Based on these power model, we further construct the process-level energy model.
- In addition, we implement the pTopW on Windows platform. This tool supplies two convenient user interfaces, which are APIs and performance counter, for power-aware applications.
- Finally, we propose EnergyGuard, a module for process-level power management.

The rest of the paper is organized as follows: Section II describes several works that are most related to our work. In section III, we introduce the energy model we proposed, describe the architecture of pTopW and the implementation of the energy models. Also, we evaluate the energy model in section. Then, in Section IV we introduce the design and implementation of EnergyGuard. We test the usability of EnergyGuard with several usage scenarios. Finally, we draw the conclusion and talk about the future work in section V.

II. Related Work

In this section, we only describe two areas, software-based power profiling and operating system-level power-aware strategies, which are most related to our work.

A. Power Profiling

As the foundation of power-aware system design, a lot of methods, which include hardware-based methods and software-based methods, have been proposed. Software-based methods are much more flexible than hardware-based methods because they do not need any devices to do power measurements. Furthermore, software-based methods can be used to estimate more fine-grained power information. We do not only estimate the power of the system level and the component level, but also could estimate the power of process, thread [13], [14] and virtual machine [15], [16]. In this section, we describe several previous publications which are related to our work, and make comparisons with them.

1) HPC-based Power Model: Hardware performance counter (HPC) is a group of registers that are used to count hardware, software and operating system events. Thus, a large group of works use HPC to build power model. Frank Bellosa [9] is probably the first person who proposed use HPC in able to estimate power information. After analysis, they found that four HPCs, which are integer operation, floating-point operation, second-level address strobe and memory transaction, are tightly related to the power dissipation of processor. In [17], Contreras *et al.* construct a processor power model by using four HPCs, which are, the instructions executed, the data dependencies, the instruction cache miss and the TLB (translation look aside buffer) misses. They also built a power model for memory. Similarly with [17], Bircher *et al.* also found that IPC (instructions retired per cycle) related HPCs are highly correlated with processor power. The research of Bircher *et al.* proved that processor related performance events are also highly related with the power of memory, chipset, I/O subsystem and disk [11]. They found the relationship by analyzing the propagation of hardware performance events among the whole system.

To implement a temperature-aware load balancing algorithm on Linux platform, Merkel *et al.* introduced a method to compute the energy consumption of tasks [14]. They also built a energy model with performance events, and they modified the kernel of Linux to do the sampling before and after each time slice. Using the sampled performance events value, they compute the energy consumption of the related task during the time slice. Different with them, pTopW can also estimate the energy consumption of each process. Even though sampling performance events during scheduling is a good choice, this method is not suitable for us because windows platform is not

open-source. In addition, the overhead will be very high if we use their method because pTopW needs to sample a large amount of performance events.

2) System Profile-based Power Model: Except using hardware performance counters, system profile is also globally used by researchers. As we know, the number of HPCs is limited; thus, it is not suitable to use HPCs to build fine-grained energy model. In addition, the power profiling thread may interfer with other applications which using HPCs. Therefore, we chose system profile to built the power model.

Li *et al.* estimated the power dissipation of operating system based on IPC [18]. They found that some operating system routines generate constant power, and other operating system routines, such as process scheduling and I/O operating, have higher relationship with IPC. Kansal *et al.* introduced a virtual machine level power model. They first built the power model and estimate the power for CPU, memory and disk, and then they distributed the power to each virtual machine based on the utilization. Our work also use similar method to compute the power of a process. The difference is that we are targeting laptop computers, so we also consider wireless network card. In [15], Dhiman *et al.* also described an online power prediction system for virtual machine.

In our previous work [13], we implemented the first version of pTop on Linux platform. Our second version is implemented on Windows platform. In addition, we added an energy model for memory and made some modifications to other energy models. Finally, this version do not rely on the system information supplied by other applications or kernel patches.

B. Power-aware Strategy

There are several work done about energy saving approaches in different aspects. As [19] refers, a lot of work related to energy are focusing on how to avoid waste. Scientists do many experiments and get the statistics then deduce which part consumes much energy, and how to improve it. At first, Scientists study specific hardware components of computer, in [20], the author found the power dissipation in micro-process. Then it goes to the energy model, in [21], [22][23], they built energy model to help research and they made the prove more reasonable. Nowadays, people are more interested in energy-aware software and system [24] design. As suggested, if we want to build energy-aware applications, we should know the energy situation first and adjust the applications' work correspondingly. Besides, in [25], the operation system needs to schedule the process by the energy which is a resource. Similarly, EnergyGuard also save energy by avoiding waste, but our work is mainly about the process level not the specific components, and we treat the software from the point view of operation system. EcoSystem [7] manages power as a kind of resource, and the available energy of process is limited to a budget. This way, they extend the battery life time through the control of power dissipation.

III. Application-level Power Profiling

In order to support fine-grained power-aware strategies, we must first design power models to calculate process-level energy consumption. In this section, we describe pTopW, a process power profiling tool, in detail. Even though it targets desktops and laptops, the idea can be easily transplant to high-end systems.

A. Energy Model

The most important part is energy model because it converts performance event values into energy and power. In this paper, we only consider the energy models of four main components, which are processor, memory, disk and wireless network card. The power models estimate the energy consumption of these devices during two sample interval. With the energy models, we can easily compute the average power of these components during the sample interval. Finally, we compute the energy consumption or power of each process based on the utilization of these hardware resources. Power is the dissipation rate of energy; thus the energy consumption in a time interval does not have much difference with power. In this paper, when we say a process's power, we mean that the execution of a process causes this amount of energy consumption in a unit time. The sum of all the active processes' power equals to the active power of these four devices.

1) The Requirements of Energy Model: Several previous articles [10] make accuracy as the foremost requirement. However, accuracy usually means complex and high-overhead; thus, it may not suitable to supply online power information. In [26], the authors proposed that responsiveness is also important because detecting power phase is critical for power-aware strategies. In addition to accuracy and responsiveness, we think flexibility and simplicity are also important for a power model which supplies online power information. Because pTopW estimates process level energy consumption, our power model cannot too complex. Otherwise, the overhead will be too high to be really used.

2) Component Energy Model: We only consider four main components of a computer system, even though other devices, such as motherboard, also account for a large ratio of power dissipation. During our research, we found that the current state of most devices is closely related with its power. For example, CPU frequency to a large extent decides the current power of processor. In addition, the usage of a device is also closely relevant with the energy consumption. The energy model we proposed estimates the energy consumption of a component in a time interval, which is the time span of two samples, and the power model estimate the average power during this interval.

Our processor energy model relies on the observation that the active power of CPU almost has linear relationship with clock speed, when the core voltage is stable, which is the case in most of the time. In addition, we assume that the clock frequency is stable during the procedure of a sample. As seen in equation 1, the first part of CPU energy consumption is static energy consumption, and the second part is dynamic energy consumption. P_s^c and P_{max}^c are the static power of of CPU and the maximum power of CPU; T_s and T_a^{cpu} are sample interval and CPU active time, which is the sum of system space time and user space time. We use the current

clock speed (F_c) and the maximum clock speed (F_m) to compute the dynamic power of processor. Here, α is a constant that denote the relationship of dynamic power and clock speed.

$$E_{cpu} = P_s^c T_s + \alpha(P_{max}^c - P_s^{cpu})(F_c/F_m)T_a^{cpu} \quad (1)$$

The memory energy model uses the amount of data written into (D_m^w) and read from memory (D_m^r) during a sample interval to compute the memory active time. Our experiment show that the memory read (S_m^r) and write speed (S_m^w) are slightly different, but the power of them are almost the same. First we compute the memory active time with equation 2. The we use equation 3 to calculate the energy consumption of memory. In this equation, P_s^{mem} and P_a^{mem} are the static power and active power of memory.

$$T_a^{mem} = (D_m^w \div S_m^w) + (D_m^r \div S_m^r) \quad (2)$$

$$E_{memory} = T_a^{mem} \times P_a^{mem} + (T_s - T_a) \times P_s^{mem} \quad (3)$$

Similarly to memory, the disk energy is also greatly relevant to read and write states of the device. Unlike our first version, windows platform has three counters that supply the ratio of time spent on idle (R_i^{disk}), read (R_r^{disk}) and write (R_w^{disk}) operations. In this way, we do not need to compute the time spent on each operation through using the amount of data read or write to divide the average speed of read and write operation. The disk energy model is shown as equation 4, in which P_r^{disk}, P_w^{disk} and P_i^{disk} are the power disk when it is reading, writing or idle.

$$E_{disk} = (R_r^{disk} \times P_r^{disk} + R_w^{disk} \times P_w^{disk} + R_i^{disk} \times P_i^{disk}) \times T_s \quad (4)$$

Finally, the power of wireless network card is only related with its state. Our experiments show that the idle power P_i^{nic} is much smaller than the power when sending P_s^{nic} or receiving P_r^{nic} data. We use the amount of data sent and received in a time interval to decide the state of wireless network card. When it is active, the energy consumption is calculated with equation 5. In this equation, D_s^{nic} and D_r^{nic} are the amount of data send and received. We assume that the state of wireless network card do not change during a sample interval. When wireless network card is inactive, the energy consumption is only the product of idle power (P_i^{nic})) and the sample interval, as seen in equation 6.

$$E_{nic} = ((D_s^{nic} \times P_s^{nic} + D_r^{nic} \times P_r^{nic}) \div (D_s^{nic} + D_r^{nic}) \times T_s \quad (5)$$

$$E_{nic} = P_i^{nic}) \times T_s \quad (6)$$

3) Process Energy Model: Unlike hardware devices, process is an abstract object; thus, a process does not have energy consumption. However, the execution of a process causes devices to consume more energy. Based on the utilization, we first divide the energy consumption of a component onto process that had used this component during the sample interval. For example, the energy consumption of a device is *E* and the utilization of a process on this device is *u*, then we compute the energy used by the process on this device as the product of *E* and *u*. We define the energy consumption of a process is the sum of the energy allocated from each component, shown as equation 7. After that, we can compute the average power of a process i with equation 8;

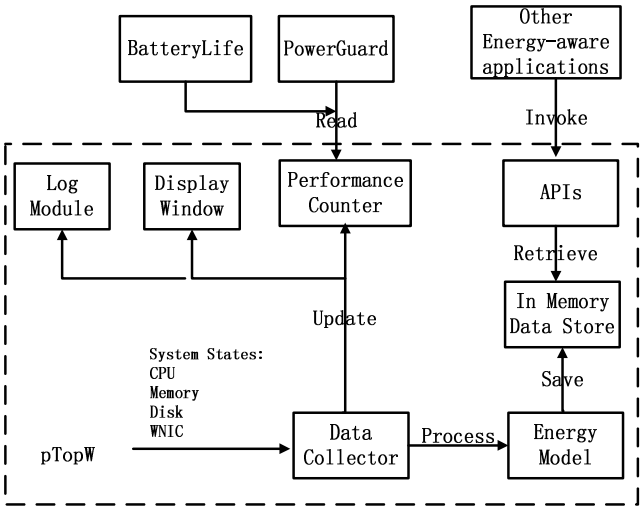

Fig. 1. The architecture of pTopW.

$$E_i = E_i^{cpu} + E_i^{mem} + E_i^{disk} + E_i^{nic} \quad (7)$$

$$P_i = E_i / T_s \quad (8)$$

We allocate dynamic CPU power to each process based on the CPU time of it. In addition, we divide dynamic power of memory based on the data input and output of the processes. The methods we used to divide disk power and network card power are similar to memory. In the next section, we will talk about the implementation of the energy model and power model in detail.

B. Design and Implementation

As shown in figure 1, the core of pTopW includes seven modules, which include data collector, energy model, in memory data store, log module, display module, performance counter and APIs. In this section, we first describe the architecture of pTopW and the function of each module. Then, we will talk about the implementation of energy module in more detail.

1) The Architecture of pTopW: Data collector, energy model and in memory data store are the most critical three modules of pTopW. Data collector is a single thread that is used to sample the system performance events. In each cycle, it first retrieves the component state information, then it reads the state information for each process. All the system data are retrieved from system performance counter APIs. Unlike our previous version, which retrieves system information from the *proc* virtual filesystem, windows supplies APIs to directly get these information. In addition, we also implemented the

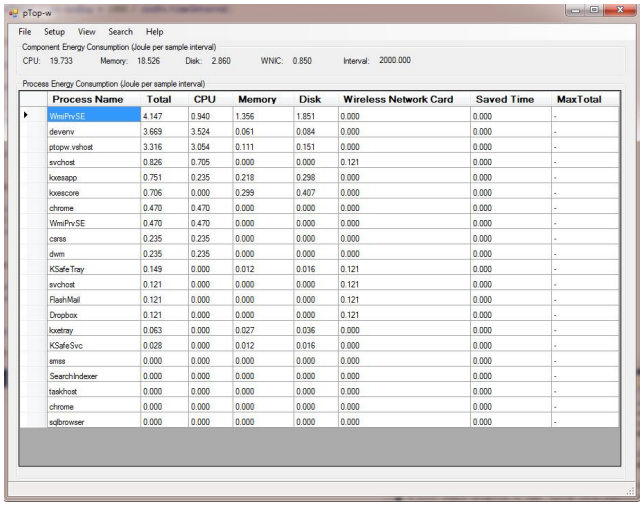

Fig. 2. The user interface of pTopW.

function of analyzing processes' network usage, which is done by analyzing the states of opened UDP and TCP connection. In our Linux version, we rely on several special kernel patches.

After a sampling, the sampled data will be sent to energy model module. Then, the energy model module converts the system state information into energy consumption and power. When the data collector receives the response, it first saves the result into the data store, and then it notifies the log module, display module and performance counter module.

The data store module saves the sampled system information and estimated energy and power information with a complex data structure, which manages data nodes in an array. The data of each sampling is saved in a data node, and in the data node we use hash table to save process related information. We design the data store in this way because the main operations on data nodes are sequential, and process related operations are key-value pair management.

Log module can save all the information into a log file, and users can use the results to do power/performance analysis. The log strategy is configurable, for example, we can configure whether process level information should be logged or not. With the data saved by the log module, users can do offline power analysis. The display module is another thread, and it refreshes the user interface on its own time interval. Data collector only pass a reference of the last updated data to display module. In this way, display module does not influence the sampling procedure of data collection module. Figure 2 is the user interface of pTopW. We list top twenty processes and the user could specify the sorting method. By default, it sorts the total power of processes.

2) Programming Interface: We supply two methods for users to get the calculated energy and power information. In the first version, we only supply APIs which could tell how much energy a process or the system has consumed in the last t seconds. In this version, we also add two functions for user to access the estimated raw data.

1. APIs

Similarly to our previous work [13], which implemented a group of APIs for users to retrieve energy consumption information, pTopW supplies similar interface to users. In addition to implementing the formatted energy consumption APIs we defined in [13], we add several new functions to retrieve the raw data. For example, we define two functions as following:

```
double[] ComponentInfo(int flag);
double[] ProcessInfo(int flag, int pid);
```

The *ComponentInfo* function returns the last estimated energy and power information of the system. The parameter *flag* is used to decide what kind of data should be returned, which maybe power, energy or both. The *ProcessInfo* function returns the last estimated energy or power information of the specified process. The parameter *pid* is the process number. With these two functions, users can get information which are asynchronous with the core of pTopW.

We use named pipe to implement the API module, which works like a network server. As soon as it receives a request, it decodes the request, processes the request and responds with the result to the client. The users can write programs while communicate with the pipe, we named it *ptopw-pipe*, we have defined directly. Moreover, a DLL, in which we encapsulate the details of communicating with the named pipe, is also available to invoke the APIs.

2. Performance Counter

We also define a group of customer performance counters to supply the realtime energy and power information. We name the category of the performance as *wsu_ptopw*. In this category, we define ten counters, which are named as *CPU Power*, *Memory Power* and so forth. Under each counter, there is a group of instances. Each instance is related to one process. The Performance counters are updated as soon as a new estimation result comes out. The users can read these counters the same with other system counters.

3) The Implementation of Energy Model: Windows platform supplies a group of APIs to access system performance data. We select the performance events that have higher relationship with the power dissipation of a device through experiment. In this section, we describe the performance events and basement values we used to build the energy model. Because we use sampling to estimate the energy consumption in a time interval, we assume the value of a performance event is stable during this time interval. Thus, the sample interval determines the accuracy of our energy model. The more short the sample interval is, the estimated result will be more accurate.

All the constant parameters of the energy model are saved in the configuration file. The users can edit these parameters based on the device of their platform. Currently, pTopW cannot automatically detect the user's hardware model because we do not have base values for these devices. That will be part of our future work.

1. CPU

The active power of processor has nearly linear relationship with clock speed when the voltage is stable [27]; thus, we use clock speed and processor time, which is the amount of time that processor used for computation, to construct the power model for CPU. When the program starts to execute, it first

collects system information, such as page size and maximum clock frequency.

In each cycle, the data collector reads the clock speed from the performance object *Win32_Processor*. In addition, the processor time is the summation of all the processes' processor time, which we can get from the *Process* object. The maximum and minimum processor power we used in the energy model is predefine in the configuration file. To make the result more accurate, users should setup these parameters based on their own platform. After calculating the energy consumption of CPU, we divide the energy consumption to each process based on the ratio of processor time.

2. Memory

Accurately estimate memory energy consumption is difficult, because memory is accessed by both processor and I/O devices. We use the amount of data which was written into or read from memory to compute the memory active time. We use three counters, which are *PagesOutputPerSec*, *PagesInputPerSec* and *CopyReadsPerSec*, to calculate memory active time. To do the calculation, we need the operation speed of these three operations, which are also predefined in the configuration file.

After we get the memory active time, we assume memory is in idle state in the rest of time. We find that the memory read and write power is almost the same, thus we only use the power of two states to calculate memory's energy consumption. Finally, the memory energy consumption is allocated to each process based on the data of I/O.

3. Disk

The power of disk is different when the disk is working at different states. The basic idea of disk energy model is find out the amount of time that disk have worked on each state in the time interval. Our last version stats the amount of bytes that disk read and write operations made, and then calculate the time with the average speed of read and write operation. However, the speed usually varies during an operation.

The disk information are retrieved from the performance data *Win32_PerfRawData_PerfDisk_PhysicalDisk*. Because this counter supplies the percent of time that disk work on each state, we do not need to use the amount of data bytes read from or write into disk in a time interval. This way, the new energy model of disk is more accurate than our first version because the speed of disk varies during read and write operations. We read three values, which are *PercentDiskReadTime*, *PercentDiskWriteTime* and *PercentIdleTime*, from this counter. Because the time spent on other states, such as searching, are not available from the system APIs, we treat these states as idle state. However, we notice that most of the time the sum of these three values are about 99 percent; thus, the neglect of this part does not influence the result too much.

We have mentioned that we use the disk usage of each process to divide the disk energy consumption into process level. We read the I/O information of each process through the function *GetProcessIoCounters*, which returns a data structure that tells the I/O information of a process.

3. Wireless Network Card

Through our experiments we found that the power of wireless network card is also greatly related with its state. The power of transmitting and receiving state is much higher than idle state. Similar to other devices, we predefined the power of wireless network card when it is in three main states into a configuration file. We read the amount of data transmitted and received from *IF_TABLE*, which is a data structure that saves network related information in windows platform. If the sum of these two values are zero, we assume the wireless network is in idle state in this interval. Otherwise, we assume it is in active states. Similarly, we divide the estimated power for each process, which is based the amount of data have been received or transmitted by the process.

C. Experiment and Evaluation

In this section, we first describe the experiment setup we used to verify the energy model. Then, we will describe how do we do the experiment and the comparison of the measured power and estimated power.

1) Experiment Setup: We evaluate our energy model on windows 7 platform. The experiment platform we used is a normal desktop computer, shown in table I. We connected several small resistors into the ATX power cables, and we measure the voltage (V_r) on the resistors. The resistor we used is only 0.03 Om; thus, it has no influence to the device because the resistor only account for no more than one percent of voltage on the line. We can measure processor power and disk power directly. However, wireless network card and memory's power cannot be divided because both of them are powered by the 3.3v voltage lines. For more information about power measurement, please read our technical report [28].

Component	Model
CPU	HP Compaq Intel Pentium 4 2.0GHzv 1 core Core Voltage 1.471V 512KB L2 Cache 8KB L1 Cache
Memory	DDR 512MB × 2 Frequency 132.9MHz Cycle Time 6 clocks
Disk	80GB Seagate Disk

TABLE I
EXPERIMENT PLATFORM.

2) Evaluation: We use applications, such as *Media Player* and *IE* to make the processor busy. Figure 3 is the comparison of the estimated power and the measured power. In this figure, we notice that our power model can estimate the trend of CPU power, even though it is not very accurate.

The higher error rate at some stages is caused by the maximum CPU power we chose. We use the average maximum power when we run several computation-intensive benchmarks, but in fact the maximum power is different when we run different benchmarks, even though all of them can make the processor work on the highest frequency.

Currently, we cannot measure the power of wireless network card or memory individually. The power we measured also includes part of motherboard circuits. We execute *IE* to test

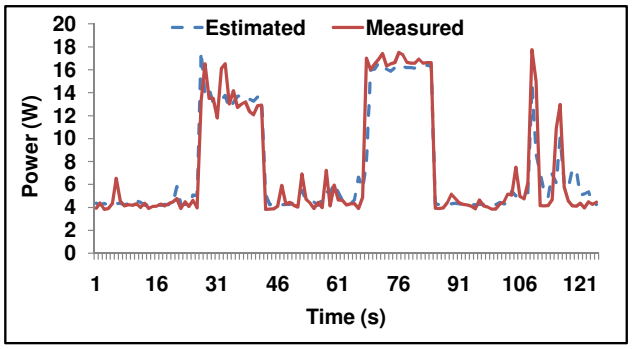

Fig. 3. The comparison of CPU power.

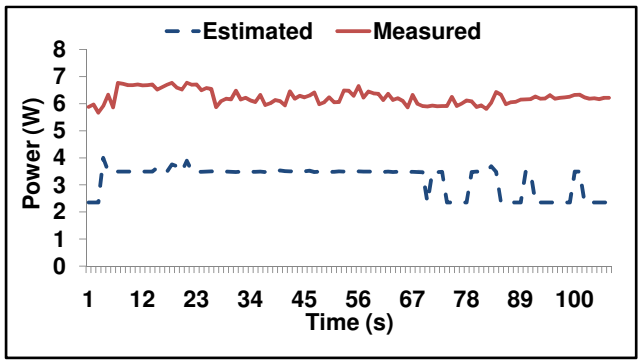

Fig. 4. The comparison of wireless network card power.

memory and wireless network card, and the result is shown in figure 4. During this experiment, we first download a file, then we open several web sites. From this figure we find that during the first stage, the estimated power is stable; however, the measured result show some fluctuation. In the second stage, the trend of the estimated power is similar with the measured power but the estimated power has slight latency when compared to the measured power.

To test the disk power model, we wrote three disk benchmarks to stress the disk usage. The first benchmark *DiskRead* generate sequential disk read operations, and *DiskWrite* benchmark generates sequential disk write operations. Another benchmark is *DiskReadWrite*, which generate disk read and write operation randomly. In the experiment we execute these benchmarks to generate the disk usage. The result is shown as figure 5. In this figure, we could see that both the measured and estimated power has four clear phases. In addition, the estimated power could show the same trend as the measured power most of the time.

During the first phase, we ran *DiskRead* benchmark to read two GB of data. In the second phase, we write one GB of data into disk, and we write three GB of data into disk during the fourth stage. The figure shows that the estimated power is very close to the measured power, and the fluctuations are also the same. In the third stage, we run the *DiskReadWrite* benchmarks. In this stage, we find that the estimated power is nearly the average of the estimated power. Furthermore, the fluctuations of the measured power do not have related fluctuations on the estimated power. The reason is because our disk power model uses three performance values to compute

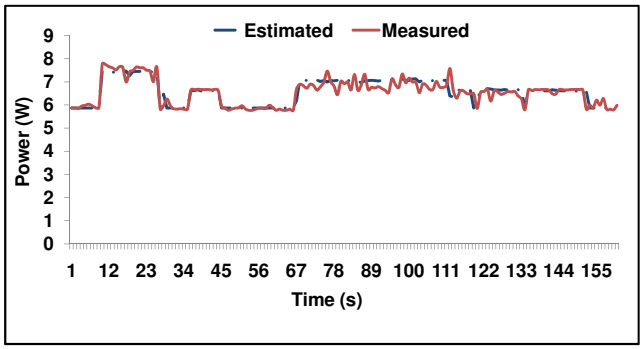

Fig. 5. The comparison of disk power.

the average disk power in the time interval.

IV. Fine-grained Power Management

Power-aware system can be designed using different angles, such as power-aware hardware design, power-aware system design and power-aware applications. Currently, a lot of power saving strategies, such as dynamic voltage and frequency scaling (DVFS) [29], have been globally used. However, most previous work use global strategies. The drawback of this kind of strategy is that performance will be significantly influenced. In addition, they are trying to decrease energy waste when the system is not active. Still a large amount of energy is wasted by redundant or even malicious processes. In addition, controlling the peak power dissipation is very important for some systems, such as temperature-aware systems. In this section, we introduce a fine-grained power-aware strategy called EnergyGuard, which uses process power profiles supplied by pTopW to make power-aware decisions. It is worth noting that pTopW and its APIs can be employed by many other process-level power management strategies as well. EnergyGuard serves both as a case study and a service here.

A. The Design of EnergyGuard

EnergyGuard has two components: monitor component and white list analysis. As describes in Figure 6, if the power of a process surpasses a threshold, which is the maximum power we acquired during the analysis process, EnergyGuard will show red background color to inform user. If a process is always highlighted, it might be in abnormal situation. White list analysis gives the rank of power consumed by all the processes. If a process, which is not in the white list, always consumes much more power than others, EnergyGuard will alert the user because the process might be harmful.

Monitor component compares the power consumption to find the suspicious process. It requires two values: threshold and current value. If the current value is higher than its threshold, we highlight the process as an alarm for users, and at the same time, we record the corresponding times. Since when most processes start to run or change from sleep mode to active mode, it will consume much power, we cannot assert a process is abnormal for surpasses threshold once. However, if the process always surpasses the threshold, there is a great

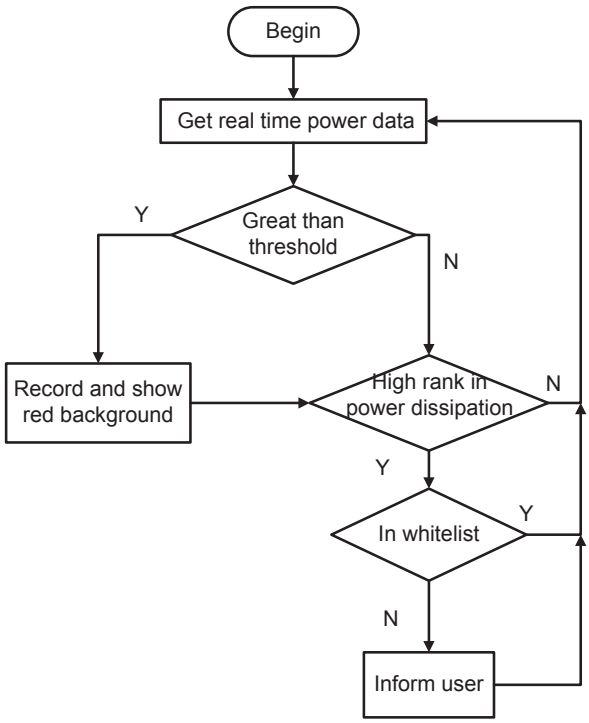

Fig. 6. The basic flow chart of the system.

change that the process is in error state. In this situation, we need to send information to user and get their attention.

We prefer to use the maximum value of k number of data we received before as the threshold. We define a suspicious or abnormal behavior as follows: for every time point t_i, we get the maximum value between t_{i-1-k} and t_{i-1} and take it as the threshold. We then compare it with the current value at time point t_i, which is achieved by using *ProcessInfo( )* interface. If the current value is larger than the maximum value, we highlight the process and record the information. Through that way, we monitor all processes run in the system and make sure they are in normal state.

Now we can estimate whether a process is in normal state or not, but it can not be used to detect malware. When we run a piece of malicious code, it may dissipate a huge amount of stable power. Hence, if we just focus on power fluctuation, this kind of applications will be ignored. In order to solve this problem, we add white list analysis. If a process consumes much more power and is not in white list, we send an information to the user to confirm its safety. For the processes in the white list, we will ignore them when we check the rank of power dissipation. No matter the process is safe or not, when we detect it for the first time, we will alarm the users, which is helpful to decrease false negative.

B. The Implementation of EnergyGuard

In our EnergyGuard prototype, we need to compare process's current value with its threshold to detect whether the process is in the normal state. We use interface *ProcessInfo( )* to return all the processes information which contains four components power and process ID at every time we call it. So, we get the real time value for all processes.

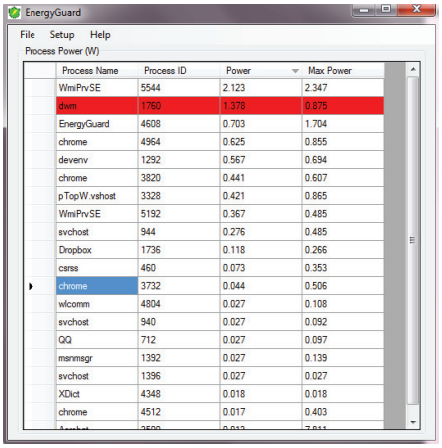

Fig. 7. The EnergyGuard application.

For the monitor part, we get the processes' power every one second from interface *ProcessInfo( )*, and store them in a hash table by using process ID as the key. Then we store the data in an ArrayList and insert the latest one to the head of the list. After we build such a power history data structure, we can easily calculate the maximum value of recent corresponding records. The user interface is shown in Figure 7, the red line points to the process whose current value is greater than the threshold. It is common that when a process starts or becomes busy, its power consumption will increase and causes the red line appear. However, we record the time stamp and the number of times that a process in "red line". If the process always greater than its threshold, we treat it as an abnormal process and report it to users.

For the white list part, we also use hash table to store the data, and the key is the same while the value is the times that the process was ranked in top five. For every five seconds we refresh the information listed to the users, and we record the ranks at the same time. If the recorded times are more than 6, we will inform the users that the power of the process is relatively higher than the others. So we jump up a dialog box to ask users if we need put the process in white list. Also, as Figure 8 shows, we have a setup menu for users to add or delete processes in the white list. Moreover, after we shut down the application, EnergyGuard will log the white list to the disk and store it for our next use.

C. Usage Scenarios

Here we use several scenarios to show how these two components work. Specifically, we examine commonly used softwares: *Microsoft Office*, *Kmplayer* (a video player), *WinRAR*, which are computationally intensive, and *PhotoShop*. Furthermore, we add some "idle" processes to make these scenarios much closer to the typical cases. These processes include *MSN* (no conversation), *Dropbox* (no file upload/download), and *Chrome* (Google personal homepage refresh).

Office: We opened *Microsoft PowerPoint 2010* to view slides, searched related topics through Internet and recorded

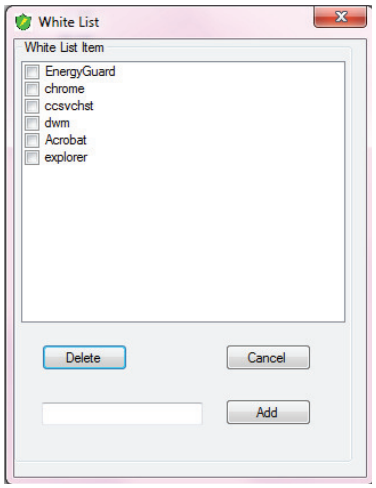

Fig. 8. The white list user interface.

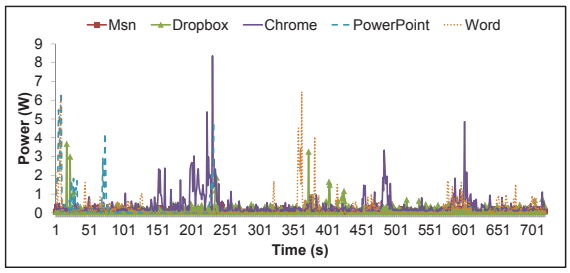

Fig. 9. The comparison of office power consumption.

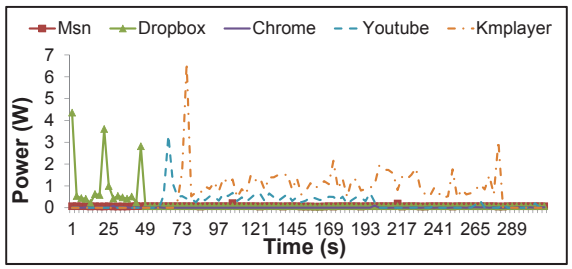

Fig. 10. The comparison of kmplayer and watching directly on internet.

them in *Microsoft Word 2010*. These operations are the basic usage patterns in our daily life. We focused on different processes during different time spans. From Figure 9, we can see that when we use *Word*, the maximum recording power could be greater than 8 Watt. While change to other applications, the minimum power of *Word* could be below 0.5 Watt.

Kmplayer: We downloaded a video which is 3 minutes and 25 seconds long. We watched it first on *Youtube*, and got the video from the browser's temp file, then use *Kmpayer* to play. Figure 10 describes the data we got, the "idle" processes are the same with the last scenario. We can see that the curve of *Kmplayer* and *Youtube* are similar, but the power of *Youtube* is less. A notable observation is the moment that we start the processes, they reach the highest power.

WinRAR: We used *WinRAR* to decompression an 2.32GB iso file. Since the decompression process is so fast, we need

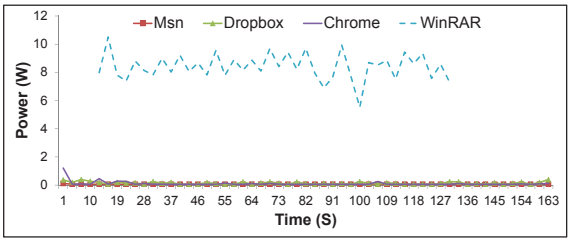

Fig. 11. The comparison of WinRAR power consumption.

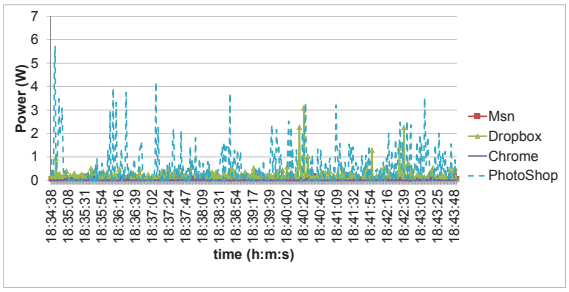

Fig. 12. The comparison of PhotoShop power consumption.

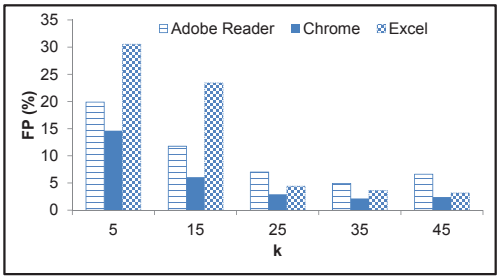

Fig. 13. An optimal value of k with respect to FP.

such a big file to get enough data as the experiments above. Through Figure 11, we know that its power range is between 7W and 10W, which is relatively higher than the others, and it is very stable.

PhotoShop: We used Adobe PhotoShop CS to open a 47.4 kilobytes PNG file and did some basic operations: add a new picture layer, write words on the picture, adjust the size and save it. Through Figure 12, we know that its power range is between 4.5 Watt and 0 Watt. Compared with WinRAR, the power distribution is kind of burst, and it has the "low" power time.

D. Performance Evaluation

The optimal value of k: We test different values of k to compare the false positive value. These experiments are done in successive time slots for the same process. The following are the work they did: *Adobe Reader* was used to read a 212 kilobytes pdf file; *Chrome* did the basic searching work; *Excel* opened a 8000 kilobytes file and executed "sum" and "if" functions. We could conclude from Figure 13 that 35 is the optimal number, which cause small false positive values and less data computation.

Malware detection: We detect malwares mainly through the white list part. If the process always appears in high rank,

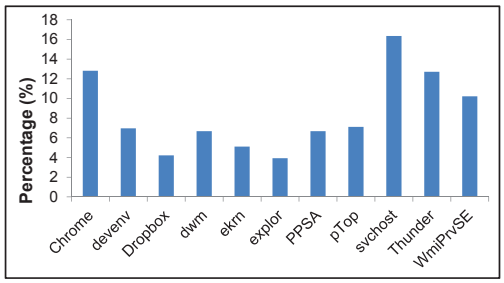

Fig. 14. The percentage of different process appears in high rank.

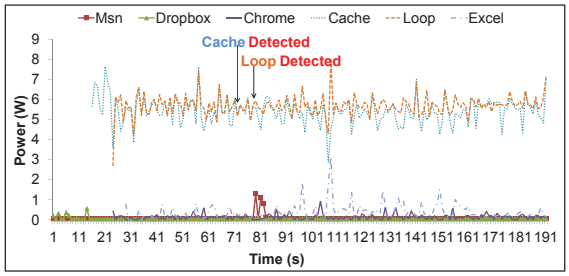

Fig. 15. Energy greedy malware detection.

we send information to the users to let them keep an eye on the process. In Figure 14, we can see the percentage of different processes appearing at high rank. Thunder is a tool used for downloading, and others are the common processes of the computer. We need to find a balance. On one hand, we need to detect malware. Thus, we can not set the number too large that malwares do not get detected. For example if we set the number is more than 18% of the total number of samples, we will not even detect Thunder which is our main process in the experiment. On the other hand, if the number is too small, the users will always be interruptted because there are so many processes that need them to confirm. Our goal is to choose the threshold which make sure most power hungry processes can be found, and ignore the common "small" ones.

We wrote two simple malwares, *Loop* and *Cache*, to do the experiment. The first one reads and writes processor cache in a dead loop, and the second one do integer computation in a dead loop. The threshold number is 10. According to Figure 15 we can see that the power of malwares is relatively stable and high. We use white list to decide whether it is harmful. An alert is sent, if the process is not in the white list.

V. Conclusion and Future Work

Power-aware operating system design requires online power information to make critical energy efficient decisions. Traditional methods are mainly focused on constructing component-level power models. However, lower level power is more important because researchers can make out additional fine-grain power saving strategies. In this paper, we first propose pTopW, which is a tool supplies realtime process level power information. The energy model we construct is based on system profiles; thus, it will nearly not be influenced by other applications and can be used as a service to supply power information to power-aware strategies. Through experiments, we found that disk and processor energy show good responsiveness. In addition, our result show that memory and wireless network energy model can show similar fluctuations with the measured result. In addition, we implemented two mechanisms for users to retrieve the power information.

Currently, the power model is retrained by the hardware information we could acquire. In the future, more details about hardware states and usage should be available in the system. In the next step, we should continue work on the energy model, especially memory and wireless network card. In addition, improving the measurement is also important for validating the models. Realtime process level power profiling needs to sample a large amount of system information. Thus, researcher should design simple energy models that have good responsiveness.

Also, we propose EnergyGuard, a fine-grained power-aware strategy. With EnergyGuard, we can find abnormal energy behaviors and notify the user. Currently, we are concentrating more on distinguishing those energy-wasting processes, but not take actions to control these processes. In the future, we will define some automatic methods to do this after we realize the user behaviors. Finally, researchers should try to find more efficient fine-grained energy saving strategies with process power profile.

References

[1] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, pp. 52–58, April 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=619062.621693

[2] C. I. Belady, A. Rawson, J. Pfleuger, and T. Cader, "The green grid data center power efficiency metrics: Power usage effectiveness and dcie," The Green Grid, Tech. Rep., 2007.

[3] L. Benini, G. De Micheli, and E. Macii, "Designing low-power circuits: practical recipes," *Circuits and Systems Magazine, IEEE*, vol. 1, no. 1, pp. 6 –25, 2001.

[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2000, pp. 83–94.

[5] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *DAC '00: Proceedings of the 37th Annual Design Automation Conference*. New York, NY, USA: ACM, 2000, pp. 340–345.

[6] A. Vahdat, A. Lebeck, and C. S. Ellis, "Every joule is precious: the case for revisiting operating system design for energy efficiency," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, ser. EW 9. New York, NY, USA: ACM, 2000, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/566726.566735

[7] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: managing energy as a first class operating system resource," *SIGPLAN Not.*, vol. 37, no. 10, pp. 123–132, 2002.

[8] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924921

[9] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, ser. EW 9. New York, NY, USA: ACM, 2000, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/566726.566736

[10] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2003, p. 93.

[11] W. L. Bircher and L. K. John, "Complete system power estimation: A trickle-down approach based on performance events," *Performance Analysis of Systems and Software, IEEE International Symmposium on*, vol. 0, pp. 158–168, 2007.

[12] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich, "Energy management in mobile devices with the cinder operating system," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 139–152. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966459

[13] T. Do, S. Rawshdeh, and W. Shi, "ptop: A process-level power profiling tool," in *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09)*, oct 2009.

[14] A. Merkel and F. Bellosa, "Balancing power consumption in multiprocessor systems," *SIGOPS Oper. Syst. Rev.*, vol. 40, pp. 403–414, April 2006. [Online]. Available: http://doi.acm.org/10.1145/1218063.1217974

[15] G. Dhiman, K. Mihic, and T. Rosing, "A system for online power prediction in virtualized environments using gaussian mixture models," in *Proceedings of the 47th ACM IEEE Design Automation Conference*. ACM Press, 2010, pp. 807–812.

[16] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 39–50. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807136

[17] G. Contreras and M. Martonosi, "Power prediction for intel xscale processors using performance monitoring unit events," in *Proceedings of IEEE/ACM International Symposium on Low Power Electronics and Design*, 2005, pp. 221–226.

[18] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 160–171, 2003.

[19] P. Ranganathan, "Recipe for efficiency: principles of power-aware computing," *Commun. ACM*, vol. 53, pp. 60–67, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721673

[20] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor," in *Proceedings of the 2004 international workshop on System level interconnect prediction*, ser. SLIP '04. New York, NY, USA: ACM, 2004, pp. 7–13. [Online]. Available: http://doi.acm.org/10.1145/966747.966750

[21] Y. Li, B. Bakkaloglu, and C. Chakrabarti, "A system level energy model and energy-quality evaluation for integrated transceiver front-ends," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 1, pp. 90 –103, jan. 2007.

[22] P. Wolkotte, G. Smit, N. Kavaldjiev, J. Becker, and J. Becker, "Energy model of networks-on-chip and a bus," in *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, nov. 2005, pp. 82 –85.

[23] Q. Wang and W. Yang, "Energy consumption model for power management in wireless sensor networks," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON '07. 4th Annual IEEE Communications Society Conference on*, june 2007, pp. 142 –151.

[24] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Commun. ACM*, vol. 53, pp. 50–58, March 2010. [Online]. Available: http://doi.acm.org/10.1145/1666420.1666438

[25] S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich, "Apprehending joule thieves with cinder," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 106–111, January 2010. [Online]. Available: http://doi.acm.org/10.1145/1672308.1672327

[26] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proceedings of the 24th ACM International Conference on Supercomputing*. ACM Press, 2010, pp. 147–158.

[27] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems : Cache, DRAM, Disk*. Denise E.M. Penrose, 2007, pp. 61–67.

[28] H. Chen, S. Wang, and W. Shi, "Where does the power go in a computer system: Experimental analysis and implications," Department of Computer Science, Wayne State University, Tech. Rep., 2010.

[29] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, ser. HPCA '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 29–. [Online]. Available: http://portal.acm.org/citation.cfm?id=874076.876477