pTop: A Process-level Power Profiling Tool

Thanh Do, Suhib Rawshdeh, and Weisong Shi
Wayne State University
{thanh, suhib, weisong}@wayne.edu

ABSTRACT

We solve the problem of estimating the amount of energy consumed by each application in the system by presenting the design and implementation of pTop, a simple and efficient process-level power profiling tool. Being a service of the operating system, pTop provides real-time information about the energy consumed by each process in terms of different resource components. pTop also supports a set of well-defined energy-aware application programming interfaces (`API`) helping the system and application developers build energy optimization policies. Different from hardware-based measurement tools, pTop is software-based, requires no additional hardware, and therefore, it is easy to apply in various platforms. Using pTop `APIs`, we developed an application adaptation scheme that extends the battery lifetime to meet certain time constraints. Experiment results showed that pTop is lightweight and easy to use.

1. INTRODUCTION

Nowadays, power management has become increasingly important in both data centers and mobile devices. Over the past five years, there has been a significant increase in the number of data centers, in addition to an estimated doubling of energy used by servers and cooling systems [1]. Moreover, the emergence of new technologies and a wide range of new mobile revolutionary products and applications have introduced a need for more power resources in mobile devices. The existing power resources are limited and still do not meet the requirements of longer battery life time. Therefore, improving energy-efficiency in both data centers and mobile devices has become more and more demanding.

In this paper, we focus on energy optimization in the application level, particularly application energy profiling. Some of the current solutions lack accuracy and exhibit some drawbacks making them difficult to use widely. First, they are hardware-based [2, 3], making it expensive, inflexible, and difficult to deploy widely. Second, they provide only offline information, therefore making it difficult for applications to adapt their behaviour at runtime. Finally, energy profiling is not offered as a service in the system, causing difficulty for system developers to implement energy-aware adaptation protocols to obtain energy optimization.

We present the design and implementation of pTop, a process-level energy profiling tool, with many helpful features. Being a service of the operating system, pTop runs at the kernel level and provides energy consumption data from all applications and processes running in the system. pTop is software-based, requires no additional hardware, therefore, easy to be applied in various platforms. Evaluation results showed that on average pTop consumes 3% of the CPU and 0.15 percent of memory. pTop is light-weight, thus, will not affect the performance of other applications. Moreover, pTop provides a set of well-defined energy-aware application programming interfaces (`APIs`) which will play an important role in making more energy real-time adapting and scheduling decisions in both mobile devices and cloud computing environments. With these `APIs`, we have developed an application adaptation framework (Section 4) for mobile devices that could extend the battery life time considerably, as well as making energy-aware QoS for application sessions across multiple domains [4]. Finally, we present future work and conclude in Section 6.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2009 ACM X-XXXXX-XX-X/XX/XX ...$5.00.

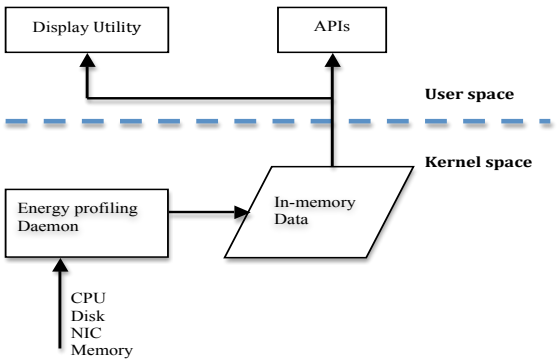

Figure 1: The architecture of pTop.

2. PTOP DESIGN

2.1 Energy model

Our tool uses a simple and feasible energy model. An application's energy consumption is estimated indirectly through its resource utilization. The quantity of energy consumed by an application E_{appi} over any time interval t can be calculated as follow:

$E_{appi} = \sum U_{ij} \times E_{resourcej} + E_{interaction}$

where U_{ij} is the usage of application i on resource j, $E_{resourcej}$ is the amount of energy consumed by resource j, and $E_{interaction}$ is the indirect amount of energy consumed by the application because of the interaction among system resources, in the time interval t.

Energy consumption of a particular resource is a function of its states(read, write, etc) and transitions. Given a particular resource, (e.g. The CPU, memory, network interface, etc), by knowing its set of states S and transitions T, its energy consumption in a time interval t is estimated as followed:

$$E_{resourcej} = \sum_{jinS} P_j t_j + \sum_{kinT} n_k E_k$$

We believe that this approach is general enough and can easily be applied to different resources. Estimating $E_{interaction}$ is more difficult, since this energy is often marked by system-level policies such as caching and buffering. We left this for future work.

2.2 System architecture

The system architecture of pTop is illustrated in Figure 1. pTop consists of an energy profiling daemon running in the background, continuously profiling resource utilization of each application process. It also tracks statistic data of states and transitions of each system resource if possible. The amount of energy consumed by each application in each time interval t is then calculated and stored temporarily in memory. Running at the kernel space, the energy profiling daemon is able to track all available system activity information.

The second component of pTop is a display utility. This component is similar to that of `top` utility, except that it provides additional information about the break-down energy consumed by the application on different resources such as the CPU, network interface, memory and disk drive. Users of battery-based mobile devices such as PDAs and laptops can use this utility to figure out and kill unnecessary and energy-consuming applications, in order to save battery energy for other more important ones.

The last but most interesting component of pTop is a set of well-defined energy-aware (`APIs`). Any process can call these `APIs` to acquire the energy consumption in terms of different energy consuming components in the last specified time period. The definition of these `APIs` is articulated next.

2.3 APIs

In this section, we define a set of energy-aware `APIs` that provide current and previous energy consumption information of each application in terms of different resources such as the CPU, network interface, memory, hard disk and display. We envision that these primitive and general `APIs` are very useful for system developers to write energy optimization middleware such as energy-aware application adapters and energy-aware schedulers. For example, the following `API` defines the interface through which an application can query the energy consumed by the CPU in a time interval.

```
int CPUEnergy( int PID, int length)
```

The inputs consist of process id (PID) and the length of time in seconds (length) back from the time when this `API` is called. The output is the energy consumed by the CPU to serve the process in terms of Joule. We should set the threshold of length parameter appropriately, depending on the amount of available memory. pTop only keeps an energy profile of running processes for the last T seconds. If a long term energy profile is needed, it is the application's task to sample every T seconds to keep the record. The `APIs` for other resources are similar by replacing the "CPU" with their corresponding names. Based on these primitive and general `APIs`, developers can write more complex and specific ones.

3. IMPLEMENTATION

We implemented pTop using C++ programming language in Linux (i.e., Fedora Core 10, kernel 2.6.28),

and we believe our ideas can be implemented in other operating systems easily. The energy profiling daemon runs in the kernel space to guarantee sufficient privileges to access to data of system activities. Like `top` utility, this daemon maintains a dynamic list of running processes, which contains information about each process' resource utilization. This data is obtained by accessing the "/proc" directory. Ideally, the operating system and hardware vendor should provide interfaces to access power consumption data of hardware at different states. Unfortunately, such interfaces are not widely available; therefore, pTop has a configuration file specifying power data from hardware vendors. In the next subsection below, due to space limit, we briefly present how we estimate the energy consumed by each of most energy-consuming resources, namely the CPU, wireless card and hard disk. Note that we are currently working on the memory energy consumption, which will be included in pTop soon.

3.1 CPU energy

We estimate the processor energy consumption by tracking the amount of time the processor running at different frequencies and the number of transitions between different frequencies in each sampling interval. This information is available in the interface "/sys/devices/system/cpu/". Based on this data, total energy consumed by the processor during the sampling interval T is calculated as follows:

$E_{CPU} = \sum_j P_j t_j + \sum_k n_k E_k$

where P_j and t_j are the power consumption and the time the processor running at a particular frequency, respectively; n_k is the number of times transition k occurs, and E_k is corresponding energy of that transition. We attribute the total energy of the CPU proportionally to the process's total CPU time.

3.2 Network interface energy

Since a wired network interface consumes a very small amount of energy, in comparison with a wireless network interface, pTop only estimates the energy process consumes in the wireless network interface. Energy spent on the wireless network interface of a process 'i' is calculated as follows:

$E_{Neti} = t_{sendi} \times P_{send} + t_{recvi} \times P_{recv}$

where t_{sendi} and t_{recvi} are the amount of time process 'i' sends and receives packet; P_{send} and P_{recv} are the power consumptions of the wireless card at sending and receiving states. We used kernel patch from atop [5] to get information about network activities per process.

3.3 Hard disk energy

Energy spent on hard disk of a process 'i' is calculated as follows:

$E_{Diski} = t_{readi} \times P_{read} + t_{writei} \times P_{write}$

where t_{writei} and t_{readi} are the amount of time process 'i' writes to the disk and reads from the disk; P_{write} and P_{read} are the power consumptions of the disk writing and reading states. We do not consider the disk transition states caused by an application, since such information is currently not available with the operating system.

3.4 Discussions

Our energy estimation approach is based on resource consumptions of application and power specification information from hardware vendors. However, hardware vendors usually specify the Thermal Design Power (TDP), i.e., the maximum amount of power of the cooling system can dissipate. This value is not usually equal the actual maximum power consumed by the hardware component. We believe that TDP is good enough for our purpose. Moreover, we also assume that energy consumed by a hardware component is proportional to its current usage. This assumption holds for most of the computer hardware components.

We do realize that the accuracy of our approach is mainly marked by the interaction between resource activities and system policies. For example, a page fault can lead to disk access, or different sizes in the network buffer can lead to different estimation results on network performance. We plan to investigate this interaction in the future.

4. PERFORMANCE EVALUATION

In order to evaluate the accuracy of our tool, we used Watts Up Pro meter [6] to profile the energy usage on the client machine, Watts Up Pro can sample the energy usage on a client machine with approximately one time per second.

Ideally pTop accuracy can be evaluated using special hardware on the motherboard like in [3], we are currently working on this. Figure 2 shows the power usage measured using Watts Up and our pTop tool under a random workload samples taken every 10 seconds from our case study applications, pTop is proved to be fine-grained, highly responsive and accurate with less than 2 Watts median error in power measurement according to our case study.

4.1 pTop Overhead

To evaluate the overhead of pTop, we ran pTop in a laptop with the same configurations in Section 3. We used pTop to monitor resource usage info

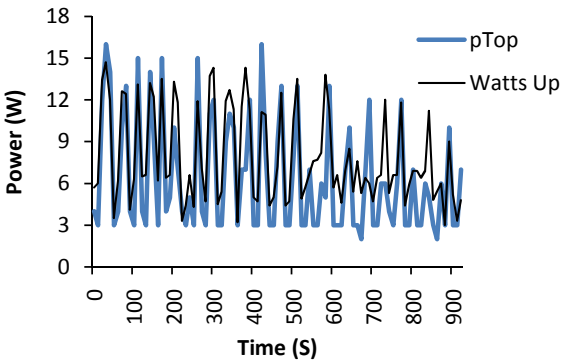

Figure 2: pTop evaluation.

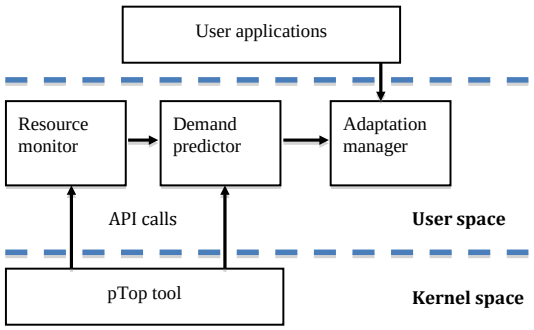

Figure 3: Adaptation framework desgin.

of all and more than 60 processes running in the systems. For each process, we monitored CPU usage, Network usage and Hard Disk usage. We set the sampling interval to 1 second. We measured the overhead of pTop using top commands. Experiment results showed that in average pTop consumes 3% of the CPU and 0.15 percent of memory. This illustrates that pTop is light weight and may not affect the performance of other applications.

4.2 Case Study

To show how important and feasible our pTop tool is, we implemented a case study experiment on a computer laptop, we used three Applications, a sorter, downloader and image viewer, the goal of the experiment is to extend battery life time to meet the time requirement of the downloader application, to download 500 MB of data before getting interrupted because of the battery's energy.

4.3 Adaptation Framework

In order to achieve our case study goal, a user-level Adaptation model that works between user applications and our energy profiling tool `APIs` has been designed and implemented. Figure 3 shows the overall framework design of our adaptation model.

Our adaptation model resides in the middle between user and kernel spaces, it consists of three main modules, *resource monitor*, *demand predictor* and *adaptation manager*. The *resource monitor* module is responsible of polling the underlying pTop `API` layer regarding power information of processes the user is running , then it forwards this information to the *demand predictor* module which uses a simple prediction algorithm to predict the power consumption of each process during the next time period. It then forwards this information to the *adaptation manager* which uses them along with applications' priority information to decide which application to be adapted and to what extent.

4.4 Experiment Setup

In this section, we implemented our case study experiment on an IBM Thinkpad T42 laptop 1.7 GHz with 1 GB of memory based on Linux kernel 2.6.28 operating system, we installed our adaptation framework and case study applications, a brief description of each application is shown in Table 1.

Each application has specific rules among the energy saving techniques available, for the sorter application the energy consumption scheme can be reduced by gradually increasing the time interval between successive sorting operations, in our experiment we set different running modes for the sorter application, those include $0, 2, 4, 8$ and up to 16 *seconds* interval between two successive sorting operations, For the image viewer application we believe that reducing the backlight brightness level of the LCD screen will decrease its energy consumption and the consumption of other applications as well. The LCD screen brightness of the IBM ThinkPad can be adapted based on 7 different modes ranging between 0% *to* 100% of brightness.

In the experiment and for ease of calculation we normalized the current battery charge capacity to $30k$ *Joules*. The experiment was stopped when this amount of energy was consumed, The experiment's goal was to allow the downloader application to finish downloading 500 MB of data during the time specified. According to the downloading speed of the wireless network at the time we ran the experiment; it would take 20 *minutes* (1200 s) to finish downloading the data from our server, this amount of time was set as the target running time we wanted to achieve. To help the downloader process finish on time, a series of adaptation decisions have been done by our framework to both the sorter application and the display brightness.

Application	Source Code Modification	Power Consumption Due	Priority	Adaptation Modes
Sorter	Yes	CPU	2	0, 2, 4, 8, 16 s intervals
Downloader	No	Network + Disk	1	-
Image Viewer	No	Disk + Display	3	0% - 100% brightness

Table 1: Applications description.

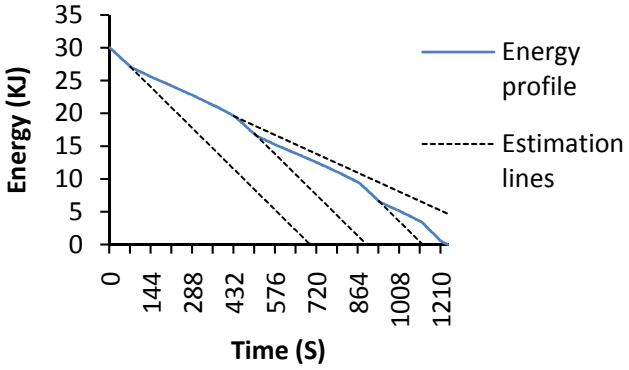

Figure 4: Adaptation decisions by time.

4.5 Experiment Results

We ran the experiment with the adaptation decisions being made every 1.2 minutes, during this time and with a sampling period of 3 seconds, the *demand predictor* module collects 24 samples of the energy consumption of each process; Figure 4 shows the sequence of adaptation decisions performed after each data sampling process. The system's target lifetime has been successfully achieved at the end.

5. RELATED WORK

A lot of energy optimization techniques have been applied so far in all three levels of computer systems. In computer architecture level, the notion of energy-proportional hardware has been proposed [7], low power hardware has also been introduced. In the system layer, a lot of techniques were introduced such as dynamic voltage scheduling, management of devices power states, workload localization and thermal-aware scheduling. And in the application level, research efforts have been focusing on techniques such as energy-adaptation by trading off application fidelity [2], controlling dynamic voltage and frequency scaling (DVFS) setting from within applications [8], energy complexity estimation, and energy-aware protocols [4, 2].

Many other studies and solutions have been done in the area of energy profiling [2, 8, 9, 10]; however, some of these solutions lack accuracy and exhibit some drawbacks making them difficult to use widely. Hardware-based energy profiling methods [2, 3] require multimeters for power sampling, multimeters are expensive, bulky and difficult to be deployed, other than that, energy profiling information is provided offline, which make it difficult for applications to adapt their behaviour at run-time. In ECOSystem [10] however, energy profiling is offered online. But we found that ECOSystem does not model clock frequency scaling, which is an important feature supported by most recent CPUs. Our profiling tool takes this into consideration and profiles the CPU energy usage based on different frequencies supported by the hardware.

6. SUMMARY AND FUTURE WORK

In this paper, we propose a process-level power profiling tool pTop, which aims to provide real-time information about the energy consumed by each process in terms of different resource components. pTop also supports a set of APIs, whose powerfulness is demonstrated by an adaptation case study. Our future work includes three main directions. First, we plan to improve the accuracy of pTop and add support for memory. Second, we believe that future operating systems should support our APIs. Finally, we intend to investigate more effective adaptation mechanisms by leveraging our APIs.

7. REFERENCES

[1] "Report to congress on server and data center energy efficiency," *Environment Protection Agency*, 2007.

[2] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," *SOSP*, 1999.

[3] Y. L. C Xian, Le Cai, "Power measurement of software programs on computers with multiple i/o components," *IEEE Transactions on Instrumentation and Measurement*, oct 2007.

[4] H. Lufei and W. Shi, "Energy-aware qos for application sessions across multiple protocol domains in mobile computing," *Comput. Netw.*, 2007.

[5] "An advanced monitor for linux-systems," *http://www.atcomputing.nl/Tools/atop/index.html*.

[6] "Watts up pro.," *http://www.wattsupmeters.com*.

[7] L. A. Barroso and U. Holzle., "The case of energy proportional computing," *IEEE Computer*, sep 2007.

[8] X. Liu, P. Shenoy, and M. D. Corner, "Chameleon: Application-level power management," *IEEE Transactions on Mobile Computing*, 2008.

[9] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," *SIGMETRICS Perform. Eval. Rev.*, 2008.

[10] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: managing energy as a first class operating system resource," *SIGPLAN Not.*, 2002.