A review of energy measurement approaches

Adel Noureddine, Romain Rouvoy, Lionel Seinturier

A Review of Energy Measurement Approaches

Adel Noureddine[1,2], Romain Rouvoy[1,2] and Lionel Seinturier[1,2,3]
[1] INRIA Lille – Nord Europe, Project-team ADAM
[2] University Lille 1 - LIFL CNRS UMR 8022, France
[3] Institut Universitaire de France
firstname.lastname@inria.fr

ABSTRACT

Reducing the energy footprint of digital devices and software is a task challenging the research in Green IT. Researches have proposed approaches for energy management, ranging from reducing usage of software and hardware, compilers optimization, to server consolidation and software migration. However, optimizing the energy consumption requires knowledge of that said consumption. In particular, measuring the energy consumption of hardware and software is an important requirement for efficient energy strategies. In this review, we outline the different categories of approaches in energy measurements, and provide insights into example of each category. We draw recommendations from our review on requirements on how to efficiently measure energy consumption of devices and software.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Review, Survey

Keywords

Energy Measurement, Energy Management, Energy Metrics

1. INTRODUCTION

Managing energy at any system level while providing a minimum accuracy requires measuring the energy available and consumed. In particular, monitoring or estimating the energy and/or resources consumption of hardware and software is a *sine qua non* condition for energy management at a higher level. This understanding of energy is however rudimentary [23], but also depends on hardware, software and execution context. New power models taking into account both computation and power management are therefore needed. Ultimately, systems should be designed to be energy adaptive (*e.g.*, being able to adapt their behavior depending on energy concerns) not just energy efficient [23].

Distributed systems add an additional layer of complexity in measuring energy. Energy efficiency can be improved by considering the end-to-end energy use of a task in all involved systems [29]. Metrics should take into consideration that energy consumption affects and is affected by other factors (such as reliability, performance). Therefore, new metrics, models and new measurement techniques are needed to support scientific evaluation of end-to-end energy management [29].

Monitoring energy consumption of hardware components usually requires a hardware investment, like a multimeter or a specialized integrated circuit. For example in [30], the energy management and preprocessing capabilities is integrated in a dedicated ASIC (*Application Specific Integrated Circuit*). It continuously monitors the energy levels and performs power scheduling for the platform. However, this method has the main drawback of being difficult to upgrade to newer and more precise monitoring and it requires that the hardware component be built with the dedicated ASIC, thus making any evolution impossible without replacing the whole hardware.

On the other hand, an external monitoring device provides the same accuracy as ASIC circuits and does not prohibit energy monitoring evolutions. Devices, such as AlertMe Smart Energy [13], monitor home devices and allow users to visualize their energy consumption history through application services, such as the now defunct Google Powermeter [21].

The previous monitoring approaches allow getting energy measures about hardware components only. However, knowing the energy consumption of software services and components requires an estimation of that consumption. This estimation is based on calculation formulas as in [40] and [22].

In this paper, we outline the concepts of state-of-the-art approaches and tools for monitoring and estimating energy consumption of software.

2. ENERGY MODELING

Estimating the energy consumption of hardware and software is often achieved through modeling resource usage for energy information. In this section, we outline the main approaches for energy models at software and middleware layers.

2.1 Energy Cost of Software

In [40, 41], the authors propose formulas to compute the energy cost of a software component as the sum of its computational and communication energy costs. For a Java application running in a virtual machine, the authors take into account the cost of the virtual machine and eventually the cost of the called OS routines. The energy cost of a software component is calculated based on the following formula:

$$E_{component} = E_{computational} + E_{communication} \quad (1)$$
$$+ E_{infrastructure} \quad (2)$$

where $E_{computational}$ is the computational cost (*i.e.*, CPU processing, memory access, I/O operations), $E_{communication}$ is the cost of exchanging data over the network, and $E_{infrastructure}$ is the additional cost incurred by the OS and runtime platform (*e.g.*, Java VM).

More specifically, the *computational energy cost* of a component is determined as the computational energy cost of its interfaces (in component-based software engineering sense). The latter is calculated as the aggregation of the energy costs of execution its bytecodes, native methods and the cost of threads synchronization (via a monitor mechanism in the Java Virtual Machine). *Communication energy cost* is calculated based on the size of transmitted and received data while accounting for the cost of transmission/receiving a unit of data. The authors rely on previous research [18, 45] to assert their argumentation that *the energy consumption of wireless communication is directly proportional to the size of transmitted and received data* [41]. Finally, *infrastructure energy overhead cost* is calculated as the energy cost of the garbage collector thread, process scheduling, context switching, and paging.

In [22], the authors take into account the cost of the *wait* and *idle* states of the application (*e.g.*, an application consumes energy when waiting for a message on the network). The following model is proposed:

$$E_{App} = E_{Active} + E_{Wait} + E_{Idle} \quad (3)$$

where E_{Active} is the energy cost of running the application and the underlying system software, E_{Wait} is the energy spent in wait states (when a subsystem is powered up while the application is using another), and E_{Idle} is the energy spent while the system is in idle state. This general model is also derived for the CPU using the following formula:

$$E_{CPU} = \{p_{Active} \times f_{Active} + P_{Idle} \times (1 - f_{Active})\} \times T \quad (4)$$

where P_{Active} is the power consumption of the CPU in active mode, P_{Idle} is the power consumption in idle mode, f_{Active} represents the CPU percentage time spent on running the application, T is the time spent running the application workload.

In [43], the authors use sensors between the power source and the system in order to measure its energy consumption. The sensors capture at regular intervals the power line conditions, such as voltage and current. The captured information is then stored in a central data collection server. After application execution, the readings from both, sensors and application, are correlated and energy consumption is estimated using the following power model:

$$E = \int_T P_S dt - P_I T \quad (5)$$

where P_S is the instantaneous power profile of the system, T is the execution time and P_I is the idle power of the system. The idle power is calculated when the system is idle while a minimum number of applications is running.

2.2 Energy aware middleware

In addition to the previous approaches, Petre [34] proposes an energy-aware model for the MIDAS middleware platform language [36]. The author proposes to model energy-awareness using the MIDAS middleware platform language [36]. MIDAS is a resource-centric language based on a previous framework developed also by the author [35] for location-aware computing. The framework defines a language for topological action systems, which is used for resource notation. The language assists the network manager on issues like resource accessibility and mobility, replicated resources and node failure and maintenance.

The author models data resources, code resources, and computation unit resources (a combination of data and code). A resource is defined as a unit that has a location and other properties. The location of these resources is modeled as a node of a network. The author distinguishes two networks: the electricity network containing the electricity sockets (modeled as electricity resources or energy supply); and the resource network of devices and resources.

In this model, energy is defined as a quantity that is consumed by the hardware devices (and indirectly by the software). The author considers that data resources and their storage do not consume energy. However, writing and reading data do consume energy. Code resources, on the other hand, need hardware to execute on. And hardware needs a power supply to work. Therefore, the author distinguishes three computation units: software or code (the unit that requests energy to run), hardware (the unit that consumes energy in order to execute the code), and electrical socket (the energy provider).

The author gives an example scenario of a user walking in a city with a mobile phone and interacting with context and location-aware elements: a statue that sends multimedia information about it, and a restaurant that sends an SMS about its menu and price. The example is modeled using the energy-aware additions to the MIDAS language. For example, the phone will have energy and functionality variables, as well as action modeling for charging using the electric sockets. Actions to apply when receiving an SMS or a video message are also included in the phone modeling.

Adding energy awareness in the MIDAS middleware platform language allows a uniform approach to modeling resources in a network. This is done by having energy modeled using the same formalism of network nodes, location or other properties. However, this modeling does not offer tools to optimize or reduce the energy consumption directly. Instead, it provides a modeling infrastructure that helps in managing energy-aware applications and networks.

2.3 Energy consumption estimation based on workload in servers

In [25], the authors propose a model for estimating the energy consumption of servers. For that, they use hardware performance counters (collected through software and operating system tools), and experimental results. A linear regression model is also proposed for predicting the energy consumption of computer jobs.

In particular, the total energy consumed by the system for a given workload is calculated using the following combined model:

$$E_{system} = \alpha_0(E_{proc} + E_{mem}) + \alpha_1 E_{em} + \alpha_2 E_{board} + \alpha_3 E_{hdd} \quad (6)$$

where α_0, α_1, α_2, and α_3 are unknown constants that will be determined using experimental results on a given server architecture (*e.g.*, linear regression analysis).

E_{proc} is the energy consumed by the processor, E_{mem} the energy consumed by the DRAM memory, E_{em} the electromechanical energy, E_{board} the energy consumed by the support chipsets, and E_{hdd} is the energy consumed by the hard drive while operating.

The energy consumption of these resources is calculated using resource-specific model. For example, the energy consumption of the hard disk is the sum of the power required to spin the disk, the idle power, and the power to read and write data. The model is thus represented using the following formula:

$$E_{hdd} = P_{spin-up} \times t_{su} + P_{read} \sum N_r \times t_r \quad (7)$$

$$+ P_{write} \sum N_w \times t_w + \sum P_{idle} \times t_{idle} \quad (8)$$

where $P_{spin-up}$ is the power required to spin-up the disk from 0 to full rotation, t_{su} is the time required to achieve spin-up, P_{idle} the power consumed by the disk when in idle, P_{read} and P_{write} are the power consumed per kilobyte of data read and write from the disk, and N_r the number of kilobyte read or written.

E_{mem} is calculated using a combination of the counts of highest level cache misses in the processor combined with the read/write power and the DRAM memory activation power. E_{em} is calculated based on the energy consumed by the cooling fans and the optical drives. E_{board} uses probe based measurements to calculate the energy required by the support chipsets. Finally, the processor energy E_{proc} is calculated as a function of its workload. The workload manifests by the CPU core temperature and the ambient system temperature. The temperature is measured using *ipmitool* [6] through sensors in the path of the outgoing airflow from the processor.

3. ENERGY MEASUREMENT & ESTIMATION

Managing and optimizing energy consumption in software while providing a minimum accuracy requires measuring the energy available and consumed. In particular, monitoring or estimating the energy and/or resources consumption of hardware and software is a *sine qua non* condition for energy management at a finer grain. In this section, we review the main approaches and tools of measuring and estimating the energy consumption of Software.

3.1 PowerScope

In [19], the authors propose a tool, *PowerScope*, for profiling energy usages of applications. This tool uses a digital multimeter to sample the energy consumption and a separate computer to control the multimeter and to store the collected data. PowerScope can sample the energy usage by process. This sampling is more precise than energy estimation, although it still needs a hardware investment.

In particular, *PowerScope* maps energy consumption to program structure. It can therefore determine the energy consumed by a specific process, and even down to the energy consumption of different procedures within the process. The implementation of the tool uses statistical sampling of both the power consumption and the system activity. The tool generates an energy profile that is analyzed later offline. Thus, the tool has no profiling overhead, but with the price of no online values. During the sampling, a multimeter is used to sample the current drawn of the profiled computer. A separate computer is also used to store the collected information and controls the multimeter (although this can also be done on the same computer).

In more details, *PowerScope* uses three software components:

1. a *System Monitor* that samples system activity by using a user-level daemon and OS kernel's modifications. The monitor records the value of the program counter (PC) and the process identifier (PID). It also records, through instrumentation, additional system information such as the pathname associated with executing processes, or the loading of shared libraries.

2. an *Energy Monitor* that runs on a separate machine and collects current samples from the multimeter. The latter transmits asynchronously the current samples to the monitor where they will be stored.

3. and an *Energy Analyzer* that uses the collected data by the system monitor and the energy monitor to generate the energy profile of the system activity. Energy usage is calculated using the formula in equation 9, and the analyzer then generates a summary of energy usage per process. The analyzing process is done offline after the execution of the program.

$$E \approx V_{meas} \sum_{t=0}^{n} I_t \Delta t \quad (9)$$

where E is the total energy over n samples using a single measures voltage value V_{meas}, I_t is the current and Δt is the interval of time.

Using their tools on adaptive video scenarios, the authors managed to obtain a 46% reduction in total energy consumption when applying video compression, smaller display size, network and disk power optimizations. However, the tool is relatively old *i.e.* 1999. Many modern hardware, operating system and software energy management techniques were not yet implemented more than a decade ago. On a modern system the energy reduction may be lower than the number the authors got in their research.

3.2 pTop

pTop [16] is a process-level power profiling tool. Similar to the GNU/Linux *top* program [28], the tool provides the power consumption (in Joules) of the running processes. For each process, it gives the power consumption of the CPU, the network interface, the computer memory and the hard disk. The tool consists in a daemon running in the kernel space and continuously profiling resource utilization of each process. It obtains these information by accessing the */proc* directory. For the CPU, it also uses Thermal Design Power (TDP) – which is the maximum amount of power the cooling system is required to dissipate – provided by constructors in the energy consumption calculations. It then calculates the amount of energy consumed by each application in a *t* interval of time. It also consists of a display utility similar to the Linux *top* utility. A Windows version is also available, so called *pTopW*, and offers similar functionalities, but using Windows APIs.

pTop's energy model is a sum of the energy consumed by individual resources in addition to energy consumed by the interaction of these resources. The following formula presents the energy consumed by an application E_{appi}:

$$E_{appi} = \sum U_{ij} \times E_{resourcej} + E_{interaction} \quad (10)$$

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.

The authors also propose a general model for energy consumption of a particular resource. The model is a function of the states (*e.g.*, read, write) and transitions of the resource. The formula is as follows:

$$E_{resourcej} = \sum_{jinS} P_j t_j + \sum_{kinT} n_k E_k \quad (11)$$

where S defines the states of the resource j, T its transitions, P_j the power consumed by resource j in a time interval t, n_k the number of transitions k, and E_k is the energy consumed by this transition.

From the general model, resources specific models can be generated. For the CPU the formula is therefore:

$$E_{CPU} = \sum_j P_j t_j + \sum_k n_k E_k \tag{12}$$

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 the corresponding energy of that transition. This calculation is based on an assumption that CPU energy is proportional to the process's CPU time.

For the network interface:

$$E_{Neti} = t_{sendi} \times P_{send} + t_{recvi} \times P_{recv} \tag{13}$$

where t_{sendi} and t_{recvi} are the amount of time process i sends and receives packets, P_{send} and P_{recv} are the power consumption of the wireless card at sending and receiving states.

For the hard disk:

$$E_{Diski} = t_{readi} \times P_{read} + t_{writei} \times P_{write} \tag{14}$$

where t_{readi} and t_{writei} are the amount of time process i writes to the disk and reads from the disk, P_{read} and P_{write} are the power consumption of the disk writing and reading states.

The authors tested their model using their process-level profiling tool. The average median error is less than 2 Watts when compared to direct energy values by a wattmeter (in their case a Watts Up Pro meter [12]) in a random workload sample taken every 10 seconds. The tool's overhead is relatively low, although not negligible, at 3% of the CPU and 0.15% of memory in a 1 second sampling interval of more than 60 processes running in the systems.

3.3 Jalen & PowerAPI

PowerAPI [14, 32] is an application programming interface (API) to monitor the energy consumption of applications, in real time, at the granularity of system processes. Jalen [31] is a software-level profiling architecture, built to monitor the energy consumption of applications at the granularity of software code (*e.g.*, methods).

Both tools uses power models for estimating the energy consumption of processes and software blocks of code. The models, thus the estimations, are divided by hardware resources and by software applications. Concretely, PowerAPI can estimate the energy consumption of a running process for the CPU, or for the hard disk, or for both or more hardware resources. Jalen can also offer per hardware resource estimations. In addition, it estimates the consumption of software at a finer grain: at the order of methods.

In particular, the CPU model is based on the standard CMOS[1] equation[37]:

$$Power^{f,v}_{CPU} = c \times f \times V^2 \tag{15}$$

where f is the CPU frequency, V its voltage, and c a constant value depending on the hardware materials (such as the capacitance and the activity factor).

The CPU model for a software process is defined as the average of the CPU power of each frequency balanced by the CPU time of all frequencies:

$$P_{comp} = \frac{\sum_{f \in frequencies} P^f_{comp} \times t^f_{CPU}}{\sum_{f \in frequencies} t^f_{CPU}} \tag{16}$$

And finally, the CPU model for a method in an application is related to the CPU time used for the execution of the method. The equation is abstracted as:

$$Power^{CPU}_{method} = \frac{Time^{CPU}_{method} \times Power^{CPU}_{thread}}{Duration_{cycle}} \tag{17}$$

where $Duration_{cycle}$ is the duration of the monitoring cycle.

The disk model follows a similar trend basing the calculations on the number of bytes read and written to the primary disk from a specific process. The equation is as follows:

$$Power^{disk}_{process} = Bytes_{read} \times Power_{reading} + Bytes_{write} \times Power_{writing}$$

where $Byte_{read/write}$ is the number of bytes read/written to the disk by the process, and $Power_{reading/writing}$ is the power required to read/write one byte from/to the disk. The latter values are specific to the hardware and therefore provided by constructors.

For methods, the power consumed is related to the size of the exchanged data with the disk. The equation is defined as a cross-multiplication:

$$Power^{disk}_{method} = \frac{Bytes^{disk}_{method} \times Power^{disk}_{process}}{Bytes^{disk}_{process}} \tag{18}$$

The network power of a process is calculated using a formula similar to the CPU power formula. From manufacturers' documentations the power consumed (in watt) for transmitting bytes for a certain duration (typically one second) according to a given throughput mode of the network card (*e.g.*, 1 MB, 10 MB), is obtained. The network power model is therefore defined as:

$$Power^{network}_{process} = \frac{\sum_{i \in states} t_i \times P_i \times d}{t_{total}} \tag{19}$$

Where P_{state} is the power consumed by the network card in the state i (provided by manufacturers), d is the duration of the monitoring cycle, and t_{total} is the total time spent in transmitting data using the network card.

The network power is calculated using the number of bytes transmitted by the application following this model:

$$Power^{Network}_{method} = \frac{Bytes_{method} \times Power^{Network}_{process}}{Bytes_{process}} \tag{20}$$

Where $Bytes_{method}$ is the number of bytes read and written by the method, $Power^{Network}_{process}$ is the power consumed by the application, and $Byte_{process}$ is the number of bytes read and written by all methods of the application.

The network power consumption per thread is therefore the sum of the network power of all methods running in the thread as shown in the following formula:

$$Power^{Network}_{thread} = \sum Power^{Network}_{methods} \tag{21}$$

The accuracy of PowerAPI is measured to a powermeter and the margin of error is calculated to vary between 0.5% up to 3%. Through PowerAPI and Jalen, the author managed to detect energy hotspot in software [31]. The given example is Jetty web server where the authors outline the energy distribution among classes and methods in their experimentation scenario.

However, the approach is limited by the implemented models (*e.g.*, CPU, disk and Ethernet network card). Jalen also uses byte-code instrumentation, therefore an non-negligible overhead is present (calculated to be around 57% for individual Tomcat's server requests). The authors, however, are developing a new version using only statistical sampling, thus reducing the overhead. They also discuss the relevancy of raw values in comparing energy consumption of software across a different pool of hardware devices.

[1]Complementary Metal Oxide Semiconductor

3.4 Other Energy Tools

In addition to the previous approaches, other tools offer energy information. Next, we present here a selection of the energy measurement tools.

3.4.1 PowerTop

PowerTop [9] is a Linux tool to *diagnose issues with power consumption and power management*. It reports an estimation of the energy consumption of software applications and system components. And also offers an interactive mode where users can apply different power management settings not enabled by default in the Linux distribution. PowerTop therefore share similarities with pTop but also limitations such as the lack of fine-grained results.

3.4.2 Energy Checker

Energy Checker [5] is an SDK made by Intel and provides function for exporting and importing counters from an application. These counters measure the time spent for a particular event or process, such as reading a file, or converting a video. The counters are then used to estimate the power consumption of the application. However, the power estimation requires a hardware powermeter, thus limiting the flexibility of the approach.

3.4.3 Joulemeter

Joulemeter [7, 39] is a software tool that estimated the energy consumption of hardware resources and software applications in a computer. It monitors resources usage, such as the CPU utilization or screen brightness, in order to estimate the energy consumption of these resources. Joulemeter uses machine specific power models for hardware configuration. Their current model takes into account processor Pstates, power utilization, disk I/O levels and whether the monitor is turned on or off. The models, however, are learned through calibration. This draws a limitation in term of flexibility as power models cannot be estimated without previous laboratory benchmarks.

3.5 Other System Tools

In addition to *pTop*, several utilities exist on Linux for resource profiling. For example, *cpufrequtils* [42], in particular *cpufreq-info* to get kernel information about the CPU (*i.e.*, frequency), and *cpufreq-set* to modify CPU settings such as the frequency. *iostat* [27] that is used to get devices' and partitions' input/output (I/O) performance information, as well as CPU statistics. Other utilities [20] also exist with similar functionalities, such as *sar*, *mpstat*, or the system monitoring applications available in Gnome, KDE or Windows. However, all of these utilities only offer raw data (*e.g.*, CPU frequency, utilized memory) and do not offer direct energy information. These raw data can, nevertheless, be used to fuel power models with information needed for estimating the energy consumption.

3.6 Application Profiling Tools

Several open-source or commercial profiling tools already propose some statistics of applications. Profilers are generally programming language dependent. GNU gprof [4] and C Profiler [3] as an example of profilers in C. For .NET languages, profilers exist such as ANTS Performance profiler [1], AQtime Pro [2] or SlimTune [10]. In Java, tools such as VisualVM [44], *Java Interactive Profiler* (JIP) [11], JProfiler [17], or the Oktech Profiler [33], offer coarse-grained information on the application and fine-grained resource utilization statistics. However, they fail in providing energy consumption information of the application at the granularity of threads or methods. For example, the profiler of VisualVM only provides self wall time (*e.g.*, time spend between the entry and exit of the method) for its instrumented methods. These tools also lack of providing network related information, such as the number of bytes transmitted by methods and thus the energy consumed.

4. DISCUSSIONS

Although many approaches exist for measuring various resources metrics, energy metrics are still lacking. Few approaches offer energy models or tools for calculating the energy consumption of software or hardware.

Energy measurement nowadays can be grouped into three categories: hardware measurement as for example in [13, 30], power models as in [22, 25, 34, 40, 43], and software measurement (as in many tools [2, 3, 4, 5, 7, 9, 10, 11, 14, 16, 17, 19, 31, 32, 33, 44]).

Hardware measurement offers high precision but at a coarse-grained level. It also requires, as it name states, additional hardware whether embedded or not. The main limitation of such approach is the inability for evolution and the difficulty to scale.

Power models provide models to calculate or estimate the energy consumption of hardware and software. Models are either too generic and coarse-grained [22, 34], or platform dependent (in particular Java) [40, 41]. Tools based on energy models suffer also from platform dependency [16, 19, 25]. The model in [25] offers a combined model to calculate the energy consumption of the system. However, their resources-specific models varies from fine-grained software-based models, such as the hard disk energy model, to coarse-grained hardware-based models, such as for the processor. The model presented uses statistical methods in their formulas, thus a tradeoff is taking place between precision and software overhead.

The most promising approach in software measurement is energy application profiling. Profilers help is understanding the system and decomposing the energy consumption of each resource. For example, in [15], the authors determined that on an Openmoko Neo Freerunner mobile phone [8], the GSM module and the display (LCD panel, touchscreen, graphics accelerator and driver, and backlight) consumes the majority of power. Still, current approaches are either coarse-grained (provide energy values at the process level) as in [16, 19], or profile some system resources without providing energy values such as [2, 3, 4, 10, 11, 17, 33, 44]. *PowerScope* [19] does not offer energy information in real time unlike *pTop* [16]. Similar to a number of other profilers, *PowerScope* collect resources information at runtime then calculates energy values offline at a later stage of the measurement. The advantage of real time solutions such as *pTop* is the ability for adaptive middleware platforms to use energy measurements for runtime energy-aware adaptations. *PowerScope* also requires hardware investment in the form of a digital multimeter while *pTop* provides similar per-process energy information using only software means.

Software profilers use software statistical sampling or software code instrumentation. Both approaches have advantages and limitation [26, 38]. Instrumentation offers two main advantages: *i) accuracy* where exact resources values are provided; and *ii) repeatability* as bytecode instrumentation produces similar results with the same environment and parameters. Sampling, on the other hand, *i)* have a lower overhead as it only occurs when sampling (unlike instrumentation where the overhead is permanent); and *ii)* does not require application source (or byte-) code modification. Although bytecode instrumentation has a non-negligible overhead for very large applications, we argue that supporting precise and accurate per-method energy profiling is better suited for diagnosing energy leaks in applications.

Table 1 presents a general comparison between the main energy measurement approaches cites in this paper.

Name	Energy Measurement	Monitored Resources	Energy Precision	Operating System Modification	Software Modification	Hardware Investment
ASIC, Powermeter	✓	Hardware Energy	Hardware	×	×	✓
OS utilities	×	Hardware and OS Resources	×, but Hardware, Software, Process for Other Resources	×	×	×
Software Profilers	×	Software Resources & Parameters	Software, Classes & Methods	Depends on Profiler	Depends on Profiler	×
PowerScope	✓	Current, Program Counter	Process, Procedure	✓	×	✓
pTop	✓	Hardware Resources (CPU, Disk, Network)	Process	× for CPU, ✓ for Network	×	×
PowerTop	✓	Hardware resources	Application	×	×	×
Energy Checker	✓	Hardware resources, Application counters	Application	×	✓ through counters	✓
JouleMeter	✓	Hardware resources	Process	×	×	×
PowerAPI	✓	Hardware resources	Process	×	×	×
Jalen	✓	Software resources	Code blocks (methods)	×	×	×

Table 1: Comparative table of energy measurement approaches

Based on our review of state-of-the-art approaches, we argue that work still need to be done for accurate and invisible energy measurement approaches. New metrics and models on both system and software levels need to be defined. These new measurements should adopt the following criteria:

1. *Accurate measurements*. Energy consumption measurement is key for energy-aware adaptations. On higher system levels (middleware and software), more accurate measurement provides better information for relevant energy management and adaptation. Measurements at a finer-granularity need to be defined, not only by providing system resources values, but rather by providing fine-grained energy consumption values for applications.

2. *Fine-grained power models*. Energy models and formulas need to be precise enough to offer energy consumption values at finer-granularity. State-of-the-art software and middleware platform models have either energy precision limitations (providing coarse-grained energy values), or provide finer-grained resources (not energy) values. We argue that finer-grained power models, without unnecessary mathematical or architectural complexity, are needed for better energy measurements.

3. *Reduce user experience impact*. Adding an additional layer of computation in order to measure energy consumption does have a non-negligible impact on user experience. Approaches implementing energy models and formulas need to be invisible for the user, the application and the underlying system. Therefore, tools need to have low or negligible overhead (in particular in term of time and energy impact). The scalability and evolution (in addition to practical usage) of the system is greatly impacted with additional hardware. Thus, no additional hardware investment needs to be used for energy measurement. Finally, measurement tools should not require the manual modification of source code of applications. We need to measure legacy or newer software without requiring the availability of their source code, or their modification by the user/developer. Instrumentation (in particular bytecode instrumentation at runtime), in this case, provides a balances tradeoff between accuracy and independence of source code modification.

4. *Software-centric approaches*. Hardware meters, although offer a precise value of the energy consumption of the device, have numerous limitations:

   - they only monitor hardware devices, not software.
   - They do not offer flexibility as it requires hardware investment.
   - The impact of energy meters on energy efficiency have also been found to decrease over time [24].

   Measuring energy consumption of devices and software is relevant when the collected information is reusable. Raw energy consumption values are hardware dependent, therefore they cannot be used *as is* in different hardware or configurations. They also may, to a lesser extend, fluctuate even on the same machine and configuration due to electro-mechanical imperfections. We argue that a software-only methodology offers enough advantages to yield this limitation of reusability, while still maintaining accuracy, fine-grained results and with little user experience impact.

5. CONCLUSION

In this review, we outlined the main approaches of energy measurements. In particular, we discussed energy metering and modeling approaches, whether they involve hardware meters of software modeling. We also outlined system level modeling approaches and tools used to estimate hardware resources usage and energy consumption.

Energy consumption measurement and estimation is a requirement for fine-grained energy optimization, but also for providing better insight of how and where the energy is being spend in software. We draw the four recommendations in our review for energy measurement requirements:

- accurate measurements for better precision in energy optimization.
- fine-grained power models for clearer insights into how and where the energy is being spent in software.
- necessity for reducing user experience impact for increasing usability and adoption of energy measurement tools.
- a need software-centric approaches for better flexibility, evolution and reusability.

References

[1] ANTS Performance profiler. http://www.red-gate.com/products/dotnet-development/ants-performance-profiler/.

[2] AQtime Pro. http://smartbear.com/products/development-tools/performance-profiling/.

[3] C Profiler. http://www.semdesigns.com/Products/ Profilers/CProfiler.html.

[4] GNU gprof. http://www.gnu.org/software/binutils/.

[5] Intel Energy Checker SDK. http://software.intel.com/en-us/articles/intel-energy-checker-sdk.

[6] IPMItool. http://ipmitool.sourceforge.net/.

[7] Joulemeter. http://research.microsoft.com/en-us/projects/joulemeter/.

[8] Openmoko Neo Freerunner. http://wiki.openmoko.org/wiki/Neo_FreeRunner.

[9] PowerTop. https://01.org/powertop/.

[10] SlimTune. http://code.google.com/p/slimtune/.

[11] The Java Interactive Profiler. http://jiprof.sourceforge.net.

[12] Watts Up Prp. http://www.wattsupmeters.com.

[13] AlertMe. http://www.alertme.com/smart_energy.

[14] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier. Powerapi: A software library to monitor the energy consumed at the process-level. In *PoweERCIM News, No. 92*, January 2013.

[15] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIX-ATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[16] T. Do, S. Rawshdeh, and W. Shi. pTop: A Process-level Power Profiling Tool. In *HotPower'09: Proceedings of the 2nd Workshop on Power Aware Computing and Systems*, Big Sky, MT, USA, october 2009.

[17] ej-techonologies. JProfiler. http://www.ej-technologies.com/products/jprofiler/overview.html.

[18] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *In IEEE Infocom*, pages 1548–1557, 2001.

[19] J. Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications (WMCSA '99)*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[20] V. Gite. How do I Find Out Linux CPU Utilization? http://www.cyberciti.biz/tips/how-do-i-find-out-linux-cpu-utilization.html.

[21] Google Powermeter. http://www.google.com/powermeter.

[22] A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. In *Proceedings of the 1st Workshop on Hot Topics in Measurement and Modeling of Computer Systems at ACM Sigmetrics (HotMetrics'08)*, pages 26–31, june 2008.

[23] K. Kant. Toward a science of power management. *Computer*, 42(9):99 –101, september 2009.

[24] C. F. Kelsey and V. M. GonzÃąlez. Understanding the use and adoption of home energy meters. In *Extended Proceedings of El Congreso Latinoamericano de la InteracciÃşn Humano-Computadora, CLIHC'09*, CLIHC'09, pages 64–71, 2009.

[25] A. Lewis, S. Ghosh, and N.-F. Tzeng. Run-time energy consumption estimation based on workload in server systems. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 4–4, Berkeley, CA, USA, 2008. USENIX Association.

[26] S. Liang and D. Viswanathan. Comprehensive profiling support in the javatm virtual machine. In *Proceedings of the 5th conference on USENIX Conference on Object-Oriented Technologies & Systems - Volume 5*, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association.

[27] Linux User's Manual. iostat. http://linux.die.net/man/1/iostat.

[28] Linux User's Manual. top. http://linux.die.net/man/1/top.

[29] Y.-H. Lu, Q. Qiu, A. R. Butt, and K. W. Cameron. End-to-end energy management. *Computer*, 44:75–77, 2011.

[30] D. McIntire, T. Stathopoulos, and W. Kaiser. ETOP: sensor network application energy profiling on the LEAP2 platform. In *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN'07)*, pages 576–577, New York, NY, USA, 2007. ACM.

[31] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. Runtime monitoring of software energy hotspots. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 160–169, New York, NY, USA, 2012. ACM.

[32] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. A preliminary study of the impact of software engineering on greenit. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 21–27, June.

[33] OKTECH-Info Kft. OKTECH Profiler. http://code.google.com/p/oktech-profiler.

[34] L. Petre. Energy-Aware Middleware. In *Proceedings of the 15th Annual International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'08)*, pages 326–334. IEEE, 2008.

[35] L. Petre, K. Sere, and M. Waldén. A topological approach to distributed computing. *Electronic Notes in Theoretical Computer Science*, 28:59–80, 2000. WDS'99, Workshop on Distributed Systems (A satellite workshop to FCT'99).

[36] L. Petre, K. Sere, and M. Waldén. A language for modeling network availability. In Z. Liu and J. He, editors, *Formal Methods and Software Engineering*, volume 4260 of *Lecture Notes in Computer Science*, pages 639–659. Springer Berlin, Heidelberg, 2006.

[37] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *MMSA'00: Proceesings of the 2nd International Symposium on Mobile Multimedia Systems and Applications*, pages 157–164, Delft, The Netherlands, 2000.

[38] O. Profiler. Sampling VS Instrumentation. http://code.google.com/p/oktech-profiler/wiki/SamplingVsInstrumentation.

[39] J. Reich, M. Goraczko, A. Kansal, and J. Padhye. Sleepless in seattle no longer. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIX-ATC'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[40] C. Seo, S. Malek, and N. Medvidovic. An energy consumption framework for distributed java-based systems. In *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering (ASE'07)*, pages 421–424, New York, NY, USA, 2007. ACM.

[41] C. Seo, S. Malek, and N. Medvidovic. Estimating the energy consumption in pervasive java-based systems. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 243–247, Washington, DC, USA, 2008. IEEE Computer Society.

[42] The Linux Kernel. cpufrequtils. http://www.kernel.org/pub/linux/utils/kernel/ cpufreq/cpufrequtils.html.

[43] A. E. Trefethen and J. Thiyagalingam. Energy-aware software: Challenges, opportunities and strategies. *Journal of Computational Science*, (0), 2013.

[44] VisualVM. http://visualvm.java.net.

[45] R. Xu, Z. Li, C. Wang, and P. Ni. Impact of data compression on energy consumption of wireless-networked handheld devices. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 302–, Washington, DC, USA, 2003. IEEE Computer Society.